Sistema de Detección de Anomalías de Red basado en el procesamiento de Payload

SISTEMAS INFORMÁTICOS

CURSO 2011-2012

Jorge Maestre Vidal
Hugo Villanúa Vega
José Ángel García Guijarro

Director
Luis Javier García Villalba

Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Madrid, Junio de 2012

Agradecimientos

Los autores agradecen la financiación que les brinda el Subprograma AVANZA COMPETITIVIDAD I+D+I del Ministerio de Industria, Turismo y Comercio (MITyC) a través del Proyecto TSI-020100-2011-165. Asimismo, los autores agradecen la financiación que les brinda el Programa de Cooperación Interuniversitaria de la Agencia Española de Cooperación Internacional para el Desarrollo (AECID), Programa PCI-AECID, a través de la Acción Integrada MAEC-AECID MEDITERRÁNEO A1/037528/11.

Abstract

Nowadays payload anomaly based detection systems go through serious difficulties when facing mimicry type attacks, as well as zero day attacks, putting protected systems on jeopardy. The system proposed on this document, as a Snort preprocessor, is based on attack instructions correlation to defend against polymorphic attacks, aditionally the use of well known attack patterns allows us to protect the network against new attacks, since a part of their code relies on already known attacks. Different ways of developement have been evaluated when pursuing these goals, being all of them presented troughout this document. While OpenMP provides us with enhaced performance on the processing of packages by using shared memory parallelism, critical sections of the processing algorithm have been improved, as well as the storage of the necessary data structures to store all of the generated information. Performance of the new implementation has been tested with real traffic from the UCM net, these results show up interesting observations about the algorithm. As current progress research lines it is important to highlight the implementation on GPGPU, CUDA or OpenCL, of critical parts of the processing algorithm, as well as the use of alert correlation systems to relieve the IDS of a part of its workload.

Keywords
palabra1, palabra2.

Resumen

Los sistemas actuales de detección de anomalías basados en la carga útil pasan por serias dificultades a la hora de defenderse frente a ataques de tipo mimicry, así como ataques día cero, pudiendo poner en serio peligro los sistemas protegidos. El sistema propuesto en este documento, como preprocesador del IDS Snort, se basa en la correlación entre instrucciones de un mismo ataque para defenderse frente a ataques polimórficos, así como en los patrones de ataques ya conocidos, pudiendo así protegerse de ataques de reciente creación, dado que basan parte de su código en algún ataque conocido. Como método para conseguir estos objetivos se han evaluado diferentes vías que se desarrollan a lo largo de este documento. OpenMP nos proporciona paralelismo en arquitecturas de memoria compartida para acelerar el procesamiento de los paquetes, mientras que se han optimizado ciertas secciones críticas del proceamiento, así como del almacenamiento de las estructuras necesarias para almacenar la información generada. Se ha evaluado el rendimiento de la nueva implementación con tráfico real proveniente de la red de la UCM, dichos resultados arrojan interesantes observaciones sobre el algoritmo. Como líneas de investigación en progreso quedaría transformar las secciones críticas del procesamiento a GPGPU, ya sea CUDA u OpenCL, así como el uso de sistemas de correlación de alertas para descargar de trabajo al IDS.

Palabras clave

NIDS, carga útil, correlación de alertas, CUDA, OpenCL, OpenMP, Snort.

Autorizamos a la Universidad Complutense de Madrid difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Autor 1

Autor 2

Autor 3

Índice General

Índice de Figuras

Capítulo 1

Introducción

El uso de internet y de las redes informáticas ha crecido en los últimos años de forma exponencial, tanto a nivel empresarial como a nivel doméstico. Sin embargo, este crecimiento del uso y las tecnologías han traído consigo un incremento aún mayor del número de ataques e intrusiones en los sistemas informáticos de todo el mundo. De ahí que la seguridad sea actualmente uno de los campos de investigación más importantes de la informática, sobre todo en el área empresarial.

El Sistema de Detección de Anomalías de Red basado en el procesamiento de Payload (SDARP) es un esfuerzo realizado por los alumnos de la Facultad de Informática de la Universidad Complutense de Madrid Jorge Maestre Vidal, Hugo Villanúa Vega y José Ángel García Guijarro como proyecto final de licenciatura dirigidos por el profesor Luis Javier García Villalba dentro del departamento ISIA (Departamento de Ingenieria del Software e Inteligencia Artificial) durante el curso 2011-2012, como continuación a un trabajo previo realizado dentro de las líneas de investigación del grupo.

SDARP es un preprocesador para el conocido sistema de detección de intrusos basado en firmas Snort. Las metas planteadas en el origen del proyecto fueron las siguientes: adaptabilidad ante amenazas tipo día cero, capacidad extracción automática de modelos de intrusión y capacidad para procesar tráfico de red en tiempo real. SDARP pretende aunar las ventajas de la detección basada en firmas con las de la detección basada en anomalías

Antes de explicar como se han afrontado las metas anteriormente expuestas es conveniente establecer ciertos conocimientos relativos al dominio de los sistemas de detección de intrusos, los cuales permitirán un mejor entendimiento del trabajo realizado por los autores de este documento en la elaboración de nuestra propia herramienta para la detección de intrusiones. Comenzaremos explicando en este mismo apartado conceptos necesarios tales como el de Seguridad Informática, amenazas, así como los distintos componentes de lo que ha venido a llamarse defensa en profundidad. Para finalizar este apartado introduciremos los sistemas de detección de intrusos así como otras técnicas avanzadas de detección de intrusiones tales como los sistemas de prevención de intrusiones y las *HoneyPots*.

1.1. Conceptos previos

1.1.1. Seguridad informática

La definición de lo que hoy conocemos como seguridad informática está basada en la obtención de: confidencialidad, integridad y disponibilidad en un sistema de información [Rus91]. La confidencialidad requiere que la información sea accesible únicamente por aquellos que estén autorizados, la integridad que la información se mantenga inalterada ante accidentes o intentos maliciosos, y disponibilidad significa que el sistema informático se mantenga trabajando sin sufrir ninguna degradación en cuanto a accesos y provea los recursos que requieran los usuarios autorizados cuando éstos los necesiten.

1.1.2. Amenazas

Un ataque es una secuencia de interacciones con el sistema que pone en riesgo cualquiera de las características arriba mencionados. En la última década el uso de sistemas de información se ha extendido entre la población, así mismo ha aumentado la exposición de los mismos ante ataques por el entorno en el que son utilizados (conexiones de acceso público, redes domesticas inadecuadamente protegidas).

Antes, los intrusos necesitaban de un conocimiento más profundo de las redes y los sistemas informáticos para poder lanzar sus ataques. Desgraciadamente, gracias al incremento del conocimiento sobre el funcionamiento de los sistemas, los intrusos están cada vez más preparados y lo que antes estaba accesible para sólo unos pocos (expertos), hoy en día cualquiera tiene herramientas accesibles con las que poder determinar las debilidades de los sistemas y explotarlas con el fin de obtener los privilegios necesarios para realizar cualquier acción dañina.

Otra fuente de riesgos proviene de los programas o aplicaciones instalados en nuestros sistemas, puesto que en la actualidad el ciclo de vida del software se ha visto drásticamente reducido. Esto es debido en parte a la adopción de metodologías de desarrollo y a la consiguiente mejora en el tiempo necesario para realizar un desarrollo, sin embargo en demasiados casos es simplemente debido a la elevada competitividad del mercado, que ha llevado a un aumento en la complejidad del software y ha obligado a reducir el tiempo destinado a diseño y pruebas del producto. Esto conlleva la presencia de errores sin detectar que posibilitan la aparición de vulnerabilidades a ser explotadas por atacantes.

1.1.3. Mecanismos de defensa

Para la correcta protección de un sistema existen diversos mecanismos tales como el cifrado, la identificación y autentificación de usuarios para proveer confidencialidad e integridad. Por otro lado existen mecanismos que velan por la disponibilidad del sistema filtrando la información, ejemplos de estos mecanismos son las listas de acceso y cortafuegos.

Como última linea de defensa de un sistema de información encontramos técnicas enfocadas a la detección de las amenazas tales como antivirus y sistemas de detección de intrusos. La característica distintiva de estas técnicas es que al contrario de las anteriores

permiten detectar intentos de romper la seguridad de un sistema.

1.2. Objeto de la investigación

El objeto de esta investigación han sido los Sistemas de Detección de Intrusos (de ahora en adelante utilizaremos sus siglas en ingles IDS para referirnos a ellos). En concreto se ha trabajado para construir un detector de intrusiones que monitorice el uso de una red y sea capaz de clasificar los eventos que en ella se producen como legitimos o pertenecientes a una intrusón por medio de su divergencia con el uso normal del sistema, el cual habrá extraido el propio detector mediante apredizaje automático.

1.2.1. Sistemas de detección de intrusos

La respuesta a esta pregunta es harto sencilla, se trata de un programa que que detecta intrusiones en un sistema. Pero esta respuesta plantea nuevas preguntas para empezar ¿Que es una intrusión?.

Se puede definir intrusión como la violación de la política de seguridad de un sistema, o como la materialización de una amenaza. Heady et al. [HEA90] definen intrusión como cualquier conjunto de acciones que tratan de comprometer la integridad, confidencialidad o disponibilidad de un recurso. Una de las definiciones más populares de intrusión es: fallo operacional maligno, inducido externamente [POWE01], aunque es bien sabido que muchas de las intrusiones proceden del interior del sistema de información.

Una vez definido que es una intrusión procedamos pues a enunciar de una manera más formal los IDS. El NIST (*National Institute of Standards and Technology*) define detección de intrusos como el proceso de monitorización de eventos que suceden en un sistema informático o red y análisis de dichos eventos en busca de signos de intrusiones. Debar, Dacier, Wespi [DEB99] han descrito un sistema detector de intrusos como un detector que procesa eventos procedentes del sistema que está protegiendo y los clasifica en pertenecientes a usos legítimos del sistema o por el contrario pertenecientes a una intrusión. El detector trabaja sobre copias de los eventos monitorizados del sistema y elimina toda información innecesaria para el análisis que va a efectuar, si una interacción es considerada como un síntoma de una intrusión, y realiza una acción en respuesta (Por ejemplo enviar una alerta al administrador del sistema).

Concepción y evolución

Las primeras investigaciones sobre detección de intrusos comienzan en 1980 en un trabajo de consultoría realizado para el gobierno norteamericano por James P. Anderson [And80], quien trató de mejorar la complejidad de la auditoría y la habilidad para la vigilancia de sistemas informáticos. Es el primero que introduce el término “amenaza” en la seguridad informática, y lo define como la potencial posibilidad de un intento deliberado de acceso a información, manipulación de la misma, o hacer que un sistema sea inutilizable. Anderson presentó la idea de que el comportamiento normal de un usuario podría caracterizarse mediante el análisis de su actividad en los registros de auditoría. De ese modo, los intentos de abusos podrían descubrirse detectando actividades anómalas que se

desviaran significativamente de ese comportamiento normal.

En 1988, en los laboratorios Lawrence Livermore de University of California en Davis, se realiza el proyecto Haystack para las fuerzas aéreas de EE.UU. Haystack era el primer IDS que analizaba los datos de auditoría y los comparaba con patrones de ataque predefinidos [Sma88]. De este modo nacía el primer sistema de detección de usos indebidos basado en firmas, el tipo de IDS más extendido en el mercado actual. En 1990, surgen los primeros proyectos de IDS basados en red. Todd Heberlein introduce tal idea y desarrolla NSM (*Network Security Monitor*) en University of California at Davis [Heb90]. En esa misma fecha, en Los Alamos National Laboratory de EEUU realizan un prototipo de un sistema experto que monitoriza la actividad de red. Su nombre es NADIR (*Network Anomaly Detector and Intrusion Reporter*) [Jac90]. A partir de este momento, comienzan una gran variedad de proyectos de investigación que hacen uso de diferentes técnicas y algoritmos para el análisis del comportamiento de un sistema informático.

En el año 2007 podemos encontrar ya más de 300 IDS comerciales cada uno con sus puntos fuertes y flaquezas, pudiendo afirmar que nos encontramos ante un conjunto de tecnologías ya maduro y plenamente aceptado por la comunidad como un elemento necesario para la defensa en profundidad de un sistema. Sin embargo ya en 2001 habían aparecido nuevos conceptos y herramientas de seguridad tales como los IPS (Sistema de Prevención de Intrusiones) o *HoneyPots* (Panales de miel) los cuales en la actualidad han desplazado parcialmente a los IDS como herramienta de detección.

1.2.2. Sistemas de prevención de intrusiones (IPS)

Los IPS derivan de los IDS, utilizan las mismas técnicas de detección, y se diferencian principalmente en que realizan el análisis sobre el tráfico real de la red, en lugar de realizarlo sobre copias del mismo, esto es denominado modo in-line. Así mismo los IPS usualmente implementan mecanismos de respuesta activa.

El utilizar un IPS en lugar de un IDS ha de ser considerado puesto que puede llegar a constituir un cuello de botella en el sistema, puesto que realizan un análisis en tiempo real de los eventos, otro riesgo añadido es la denegación de servicio por parte del IPS a usuarios legítimos del sistema. Una buena política para decidir sobre la adopción de un IPS sobre un IDS es analizar el modelo de negocio y determinar que característica primar: bien la accesibilidad (IDS) o bien la seguridad (IPS). Un IDS puede transformarse en un IPS de facto siempre que posea las siguientes propiedades: modo in-line de análisis e implementación de respuestas activas. Snort permite tanto el análisis in-line de los datos como la configuración de respuestas activas en determinados casos.

1.2.3. Panales de miel (*HoneyPots*)

En el segundo caso nos encontramos con una técnica complementaria basada en la creación de elementos artificiales, aparentemente vulnerables con el objetivo de atraer a los atacantes. Estos elementos de red no pueden ser accedidos por ningún servicio legítimo del sistema por lo que todo acceso es inmediatamente catalogado como ataque. Esto resulta útil para reunir información sobre técnicas de ataque y su procedencia las cuales

pueden ser usadas a posteriori para la mejora del resto de elementos de seguridad de la red.

1.3. Estructura del trabajo

En el presente capitulo se han introducido conceptos tales como la seguridad informatica, amenaza y posibles mecanismos de defensa. Además se ha introducido el objeto de este trabajo que son los sistemas de detección de intrusiones y proporcionando una perspectiva historica de los mismos. En los siguientes capitulos se tratará el estado del arte de los IDS (Capítulo 2). A continuación se presentará el IDS de código libre Snort sobre el que se ha integrado nuestro detector (Capítulo 3). Seguidamente mostraremos los trabajos realizados en el proyecto SDARP así como los resultados obtenidos tanto en detección de amenazas como en velocidad de procesamiento de paquetes (Capítulo 4). Por último presentaremos las conclusiones, tres propuestas de trabajo futuro que consideramos de gran interes para la mejora de nuestra aplicación y para la consecución de mejoras significativas en el campo de la detección de intrusos.

Adicionalmente encontramos los apéndices que contienen información relacionada con el proyecto pero explicable de manera atómica: preprocesadores de Snort, alertas de Snort, guia de uso de OpenMp y manual de instalación de la herramienta SDARP.

Capítulo 2

Estado del arte

En esta sección se muestran ciertos aspectos que motivan la investigación en torno al rol de los sistemas de detección de intrusos. Se da una perspectiva general de la estructura de un IDS, las técnicas que han sido utilizadas a lo largo de los años como método de detección de intrusos, y los temas de investigación más candentes de la actualidad.

2.1. Estructura general para un IDS

Debido a la gran cantidad de alternativas existentes en la implementación de IDS el dar una perspectiva estructural de los mismos resulta cuanto menos complicado. Por ello usaremos como base el trabajo realizado por importantes grupos de investigación para establecer estándares entorno a los que basar nuevos desarrollos [15].

A finales de los años 90, la *Defense Advanced Research Projects Agency* (DARPA), constituyó un grupo de investigación enfocado en el desarrollo de una propuesta de una plataforma para IDS, el *Common Intrusion Detection Framework*(CIDF) [15]. Los componentes del IDS se pueden encontrar en una misma maquina o distribuidos entre diferentes equipos con lo cual nos encontraríamos frente a un IDS centralizado o distribuido respectivamente. Otras propuestas de estandarización han sido realizadas como por ejemplo el elaborado por la ISO [16]. La Figura 1 representa los componentes y los canales de comunicación entre ellos. La división entre los componentes es funcional no física, esto quiere decir que los módulos pueden estar situados en distintas máquinas con representaciones internas de eventos muy dispares entre si por lo que se hace necesario el usar para la comunicación entre ellos un formato independiente de la misma para eso usan los mencionados objetos GIDO necesitando para ello un lenguaje que permita definirlos, en este caso se propone el *Common Intrusion Specification Language* (CISL) que tenía entre sus metas: la expresividad para definir cualquier tipo de intrusión, unívoco en caracterizar intrusiones, extensible para dar cabida a nuevas intrusiones, simplicidad en la construcción de las representaciones y portabilidad para poder ser implementado sobre una amplia variedad de plataformas [17].

- **Entorno:** Si bien no forma parte del IDS la elección del entorno monitorizado afectará al resto de elementos del mismo. Baste por el momento decir que se trata o bien del flujo de paquetes en una red, o de los registros de auditoría de los procesos que se están ejecutando en un host.

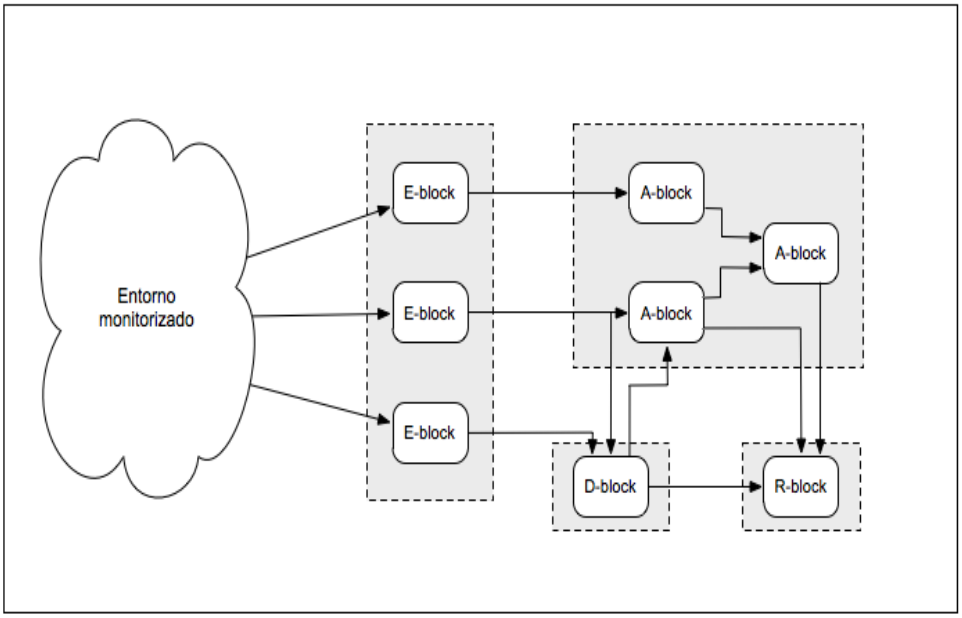

Figura 2.1: Esquema de la arquitectura CIDF

- **E-block:** bloque de escucha. Proporciona información sobre los eventos del entorno al resto de elementos del IDS. Esta información se materializa en forma de objetos de comunicación *General Intrusion Detection Object*(GIDO).

- **A-block:** motor de análisis. Son los bloques encargados de analizar los datos recogidos por los bloques de escucha y clasificarlos en una de las 4 categorías en la Figura 2.2. Los bloques de análisis reciben y envían objetos de comunicación, los objetos enviados por un módulo de análisis se presupone que sintetizan los eventos de entrada.

- **D-block:** Bloque de base de datos. Estos bloques son los encargados de almacenar objetos GIDO observados en los E-blocks para su procesamiento en los bloques de análisis y de respuesta si esto fuera necesario.

- **R-block:** motor de respuesta. Estos bloques consumen objetos GIDO. La respuesta puede ser pasiva o activa lo cual significa que de manera automática realiza acciones contra la intrusión. El empleo de respuestas activas ha de ser cuidadosamente considerado ya que pueden incurrir ante un falso positivo en una denegación de servicio a un usuario legítimo.

En el 2001 el CIDF se fusionó con el IETF para pasar a denominarse *Intrusión Detection Working Group* (IDWG) que en el año 2007 publicaron el *Intrusión Detection Message Exchange Format* (IDMEF) el cual pese a romper en parte con lo propuesto por CIDF continua proponiendo un protocolo de mensajes con características similares al ori-

ginal [18].

Pese a que el CIDF no tuvo éxito como estándar para el desarrollo de nuevos sistemas de detección de intrusos resulta satisfactorio como ejemplo genérico de los componentes de cualquier IDS y de las relaciones existentes entre ellos. Cabe destacar que en la actualidad es cada vez más necesaria la colaboración entre distintos IDS para la correcta detección de las amenazas y la minimización de las falsas alarmas por lo que la emergencia de un estándar para este tipo de sistemas resulta de un gran interés.

2.2. Clasificación de eventos y evaluación de rendimiento

Como se ha explicado anteriormente un IDS realiza su trabajo mediante la clasificación de eventos producidos en el sistema en positivos y negativos. Está clasificación se divide en cuatro categorías como se muestra en la figura siguiente [1] .

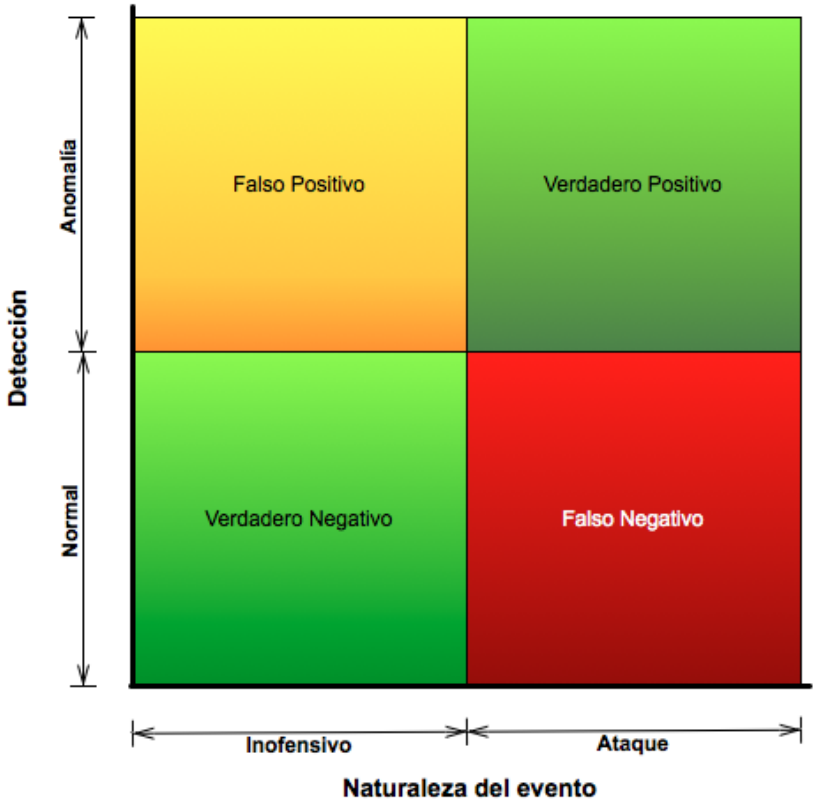

Figura 2.2: Clasificación de los Eventos analizados respecto a Naturaleza del mismo

Los conjuntos de eventos marcados en verde (Falsos positivos y verdaderos positivos) son en principio inofensivos, bien por que han sido correctamente identificados como pertenecientes a un ataque o por pertenecer a tráfico legítimo. Aquellos eventos que son clasificados de manera errónea como pertenecientes a un ataque pero en realidad pertenecen a un uso legítimo del sistema son denominados falsos positivos y están resaltados en naranja, este tipo de eventos no son deseables pero en si mismos no resultan peligrosos.

La última categoría, resaltada en rojo, constituye la más crítica de las cuatro puesto que los eventos de está categoría son aquellos que han sido erróneamente clasificados como inofensivos cuando en realidad pertenecen a un ataque.

El rendimiento de un IDS se realiza en base al ratio de falsos positivos y la tasa de detección, o lo que es lo mismo verdaderos positivos. Independientemente de la técnica de detección utilizada el detector habrá de ser calibrado este proceso podrá ser llevado a cabo manual o automáticamente pero se resume en ajustar parámetros del algoritmo empleado para ajustarlo al sistema protegido. El proceso de calibrado no es trivial y por regla general exige alcanzar un compromiso con respecto al número de falsos positivos generados. Un calibrado excesivamente restrictivo sería contraproducente con respecto a la accesibilidad del sistema impidiendo su uso legítimo, en otro lado encontramos que un calibrado laxo induce un elevado ratio de falsos negativos que como ya hemos comentado constituye nuestro peor escenario posible.

A pesar de la ausencia de marcos de evaluación común, existe un gran abanico de herramientas analíticas que pueden ser usadas para evaluar aspectos específicos de los sistemas de detección. Como ejemplo tenemos las curvas de *Receiver Operating Characteristic* (ROC) que nos permiten evaluar el rendimiento del detector. Típicamente para esto se enfrenta el número de verdaderos positivos de un sistema frente a la probabilidad de falsos negativos. El método para obtener estas curvas varía de un sistema a otro pero usualmente se consiguen modificando alguno de los parámetros del algoritmo empleado para realizar la detección.

2.2.1. Conjuntos de datos para la evaluación

A la hora de entrenar cualquier modelo de *data mining* se necesita un conjunto de datos suficientemente representativo. Una vez construido el modelo, se debe evaluar su precisión mediante la tasa de aciertos, el número de falsos positivos y el de falsos negativos. Para ello existen, disponibles en Internet, conjuntos de datos que representan la actividad de una red.

Por un lado, se pueden encontrar los DARPA *intrusion detection evaluation data sets* [3] [2], que han sido utilizados como método de evaluación en multitud de sistemas de detección de intrusos. Son conjuntos de datos que se usan para comparar los resultados de los diferentes grupos de investigación. Se dispone de dos conjuntos de datos, el de 1998 y el de 1999. Inicialmente fueron diseñados para evaluar la tasa de falsas alarmas y la tasa de detección de los diferentes IDS. Los datos fueron recogidos de una red local que simulaba una base de las fuerzas aéreas norteamericanas y contenía tanto ataques conocidos como nuevos.

El conjunto de datos del 1998 estaba compuesto por 32 tipos de ataque, recogidos durante nueve semanas de actividad (siete semanas con el fin de entrenar los algoritmos propuestos, y otras dos semanas de datos para las pruebas). En el conjunto del 1999 se lanzaron más de 300 ataques de 38 tipos, en la red que simulaba cientos de usuarios en miles de estaciones. McHugh realizó una crítica a esta evaluación, argumentando que los métodos empleados para la generación de datos no eran adecuados, ya que los datos generados de forma sintética no eran los más apropiados para simular entornos reales [4].

Consideraciones

Como puede deducirse estos conjuntos aunque usados habitualmente para la evaluación de los detectores resultan inadecuados para realizar ninguna prueba significativa a día de hoy y en la mayoría de los casos hay que proceder a elaborar conjuntos de datos de prueba ad-hoc para cada evaluación.

2.3. Taxonomía

Desde su concepción en los años 80 hasta hoy en día han surgido una gran cantidad de IDS. Podemos encontrar las principales motivaciones de este hecho en la aún más rápida evolución en la sofisticación y variedad de los ataques así como la disminución de las habilidades requeridas para llevarlos a cabo.

Debido a la variedad de alternativas, las cuales presentan como es lógico sus particulares bondades y flaquezas, se hace necesario escoger una serie de ejes (características definitorias) en torno a los que agrupar la taxonomía con el fin de poder establecer una clasificación correcta para cualquier IDS existente. A continuación presentaremos estos ejes (estrategia de respuesta, entorno monitorizado, situación de componentes y estrategia de detección). En cada una de las características analizaremos las posibles alternativas que podemos encontrar.

2.3.1. Clasificación por estrategia de respuesta

Desde el punto de vista de la respuesta del detector cuando este clasifica un evento como sospechoso existe una clara división entre los IDS pasivos que tan solo generan una alerta y los IDS activos que ejecutan contramedidas para intentar ralentizar e incluso detener un ataque contra el sistema.

Respuesta pasiva:

Típicamente los IDS presentan este tipo de respuesta siendo la respuesta activa una característica más propia de un IPS. Este tipo de respuesta consiste en enviar alertas al administrador del sistema cuando un evento supera el umbral de confianza especificado para el detector. El principal problema de este tipo de IDS es que puede sobrepasar a los operadores humanos con alertas procedentes de un mismo ataque, o simplemente con la generación de falsos positivos (En un entorno típico de trabajo de un IDS pueden llegar hasta el millón diarios).

Respuesta activa:

Es de fácil comprensión que la respuesta activa presenta ventajas fundamentales en la protección de cualquier sistema, puesto que una pronta respuesta ante un ataque significa que la integridad del sistema estará mejor protegida que se ha de esperar a que el operador humano analice las alertas generadas por el IDS y efectúe las acciones correspondientes. La naturaleza de las acciones que pueden ser llevadas a cabo por parte de un IDS en respuesta a un positivo en el análisis dependen esencialmente de si se trata de un NIDS o de un HIDS [5].

Contramedidas de red:

Existen cuatro clases de contramedidas que se pueden utilizar desde la perspectiva de un NIDS, estas medidas corresponden a las capas de la pila de protocolos TCP/IP, queda excluida por motivos obvios la capa física sobre la cual ninguna acción es aplicable por parte del IDS.

1 *Contramedidas de la capa de enlace:*
la desconexión de un puerto asociado a un sistema desde el que se está lanzando un ataque. Esto es únicamente factible si el ataque está siendo lanzado desde un sistema dentro de la red local.

2 *Contramedidas de la capa de red:*
interactuar con el cortafuegos externo o con las tablas del router para bloquear todas las comunicaciones procedentes de una IP concreta.

3 *Contramedidas de la capa de transporte:*
Se generan paquetes TCP RST para deshabilitar la sesión TCP desde la que procede el ataque. De la misma manera se enviaran paquetes ICMP de error si el ataque se está realizando mediante el protocolo UDP

4 *Contramedidas de la capa de aplicación:*
estas medidas solo son aplicables si el IDS dispone de modo in-line y este se encuentra habilitado, esto significa que está realizando el análisis sobre datos reales y no sobe copias de los mismos. Las contramedidas de la capa de aplicación consisten en alterar los payloads maliciosos para convertirlos en inofensivos antes de alcanzar su objetivo, esto requiere además modificar los códigos de comprobación de error correspondientes a estos paquetes.

Las contramedidas de la capa de enlace y de la capa de aplicación requieren de mecanismos de timeout para volver a habilitar los puertos e IP's bloqueados, mientras que los de las capas de transporte y aplicación no los precisan debido a que están vinculados a las sesiones TCP/UDP.

Contramedidas locales:

La principal ventaja de implementar contramedidas locales es el acceso a las API's y al núcleo mismo del sistema operativo (SO) así como el ya mencionado análisis de información de alto nivel que hace inútiles la mayoría de. Normalmente cuando una aplicación se encuentra comprometida realizará alguna llamada a la API del núcleo del sistema operativo por ejemplo intentando forzar un desbordamiento del *buffer*, la creación de un nuevo servidor, modificación de ficheros en disco. Las respuestas posibles tras la detección de cualquiera de estas acciones se dividen en dos categorías: modificaciones de sistema y cuñas de aplicación.

Contramedidas

1. ***Modificación del sistema:***este conjunto de técnicas está incluye la alteración de permisos en el sistema de ficheros, borrado de virus y gusanos, modificación del con-

junto de reglas del cortafuegos local.

2. ***Cuñas de aplicación:***se trata de código especifico que intercepta las interacciones de una aplicación con el SO y las inspecciona para detectar código malicioso. Para el SO la cuña no es más que otra aplicación y se encuentra sometida a las mismas restricciones de seguridad que el resto de aplicaciones del sistema.

3. ***Mecanismos proporcionados por el SO:***es conveniente que el sistema operativo protegido implemente mecanismos a nivel de núcleo para reforzar cualquier posible contramedida tomada por el IDS. Ejemplos de estos mecanismos son el *Linux Intrusion Dtetection System* (LIDS) y el *Mandatory Access Control* (MAC) también para sistemas Linux y del cual se puede encontrar una implementación de referencia en [6].

Consideraciones

Pese a constituir una mejora respecto a la simple observación de un registro de alertas, sin embargo exacerba el problema de los falsos positivos. En un IDS con respuesta pasiva los falsos positivos pueden llegar a ser muy molestos puesto que pueden llegar a sobrepasar la capacidad de los operadores humanos para discriminar entre una falsa alarma y una real [29] , sin embargo cuando se emplea la respuesta activa podemos llegar a incurrir en denegaciones de servicio arbitrarias a usuarios legítimos con lo que el rango de circunstancias en las que se emplea una respuesta activa ha de ser limitado a un pequeño conjunto de actividades maliciosas bien definidas como por ejemplo la exploración de puertos.

2.3.2. Clasificación por entorno monitorizado

Host IDS (HIDS)

Este tipo de IDS fue el primero en ser desarrollado y desplegado. Típicamente están instalados sobre el servidor principal, elementos clave o expuestos del sistema.Los HIDS operan monitorizando cambios en los registros de auditoría del host tales como: procesos del sistema, uso de UCP, acceso a ficheros, cuentas de usuario, políticas de auditoria, registro de eventos. Si los cambios en alguno de estos indicadores excede el umbral de confianza del HIDS o hace peligrar la integridad del sistema protegido el HIDS tomará una acción en respuesta. La principal ventaja de estos detectores es su relativo bajo coste computacional puesto que monitorizan información de alto nivel, sin embargo esto también constituye su principal inconveniente puesto que solo pueden lanzar una alerta una vez que el daño al sistema ya está hecho. Puede verse un esquema de un sistema protegido por un HIDS en la Figura 2.3.

Ejemplos de HIDS comerciales son: Symantec Host IDS, ISS BlackICE PC16, TCPWrappers, Enterasys Dragon Host Sensor.

Network IDS (NIDS)

NIDS son las siglas de *Network Intrusion detection System*, estos IDS analizan eventos del tráfico provenientes del tráfico de red para clasificarlos adecuadamente en pertenecien-

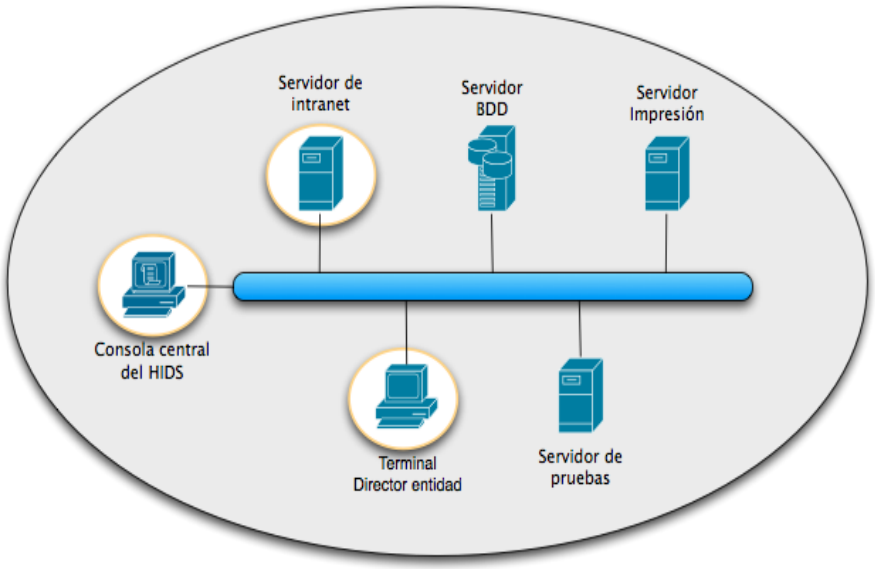

Figura 2.3: Ejemplo de sistema protegido mediante HIDS

tes o no a un ataque. Existen diversas formas de clasificar los NIDS: por objeto de análisis, por situación de los componentes del IDS, por tipo de análisis efectuado y por último por la técnica de análisis empleada. Para el propósito de este documento tan solo entraremos en detalle sobre el objeto de análisis.

Un NIDS monitoriza los eventos de red, esto significa que el objeto de análisis son los paquetes y tramas de los diversos protocolos de la misma. Estos eventos de red se dividen en dos partes: cabecera y contenedor, en consecuencia se han desarrollado NIDS's que abordan el problema de la detección de intrusiones mediante el análisis de una de estas componentes dado que cada una de estas aproximaciones trae ventajas en la detección de distintos tipos de ataques.En la figura 2.4 puede apreciarse un sistema protegido mediante un NIDS que monitoriza un segmento de red.

Análisis de cabeceras

El análisis de cabeceras se basa en analizar el flujo de paquetes teniendo en cuenta los datos de alto nivel de las cabeceras de los mismos. Por supuesto el análisis se puede realizar respecto a las cabeceras de la capa de aplicación, respecto a las cabeceras de la capa de red o incluso a las tramas de los diferentes protocolos de la capa de enlace.

Este análisis presenta un buen rendimiento en la detección de ataques tipo flood, en los que se envían una gran cantidad de paquetes. Se intentan descubrir patrones con respecto al puerto destino, protocolo utilizado y otra información presente en la cabecera, esto permite analizar gran número de paquetes a un coste relativamente bajo en cuanto a recursos. El problema para esta aproximación se da cuando se intentan descubrir ataques de tipo non-flood en los cuales la información distintiva respecto al tráfico legítimo se encuentra en el payload y no en las cabeceras.

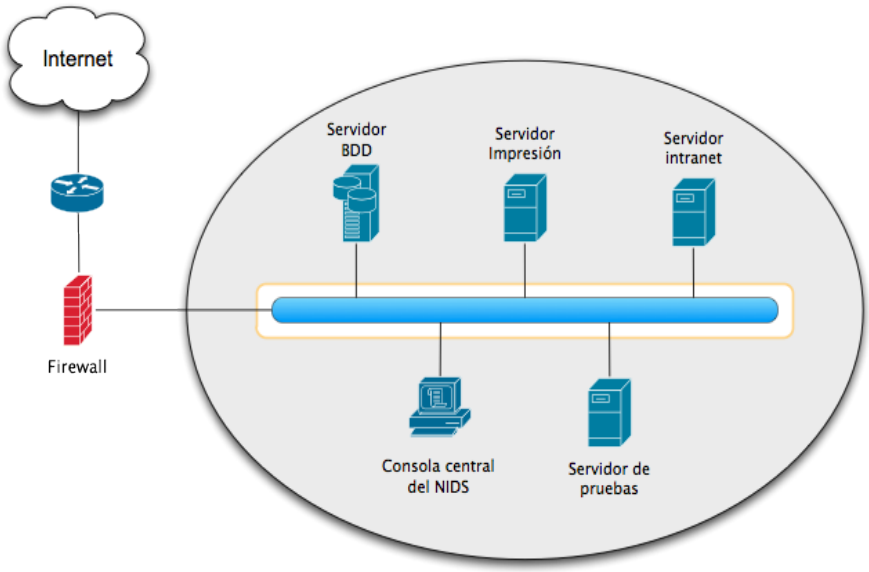

Figura 2.4: Ejemplo de sistema protegido mediante NIDS

Otro punto a tratar en este tipo de detectores es el siguiente: una de las informaciones más utilizadas por estos analizadores es el puerto de destino del paquete, siendo esto utilizado para determinar el protocolo que está utilizando el servicio. Sin embargo esto presenta una limitación importante para estos NIDS ya que frecuentemente las aplicaciones hacen uso de puertos no estándar, siendo un ejemplo de esto las aplicaciones tipo *Voice over IP* (VoIP) como Skype la cual hace esto para permitir las comunicaciones sin interferencias de los cortafuegos locales. Este mismo comportamiento es explotado por parte de los atacantes con los mismos fines siendo necesario la implementación de otros mecanismos a nivel de aplicación para determinar el protocolo utilizado.

Análisis de payload

Al analizar el payload al contrario del análisis de cabeceras se procesan datos de bajo nivel que se encuentran dentro del paquete capturado de la red monitorizada. De nuevo encontramos trabajos centrados en analizar el payload distintas maneras dependiendo del protocolo al que pertenezca el paquete, sin embargo esto presenta un problema en la adaptación a nuevos servicios. En otro sentido se han propuesto alternativas que analizan el payload independientemente del protocolo utilizado, siendo estas más costosas pero presentando una mejor adaptación ante nuevas circunstancias.

El análisis de payload ha demostrado ser útil en la detección de ataques de tipo non-flood los cuales suelen estar contenidos dentro de un solo paquete. Ejemplos de este tipo de ataques son los conocidos *User To Root* (U2R) y *Remote To Root* (R2R).

El principal problema que presenta este análisis es el coste en términos de recursos para el sistema que efectúa el análisis debido a la vectorización de los paquetes para su procesamiento. Otra dificultad añadida es la heterogeneidad en los payloads que circulan en la red.

Consideraciones sobre el uso de NIDS

Un NIDS actúa como un *sniffer* del trafico de red, capturando y descodificando paquetes que circulen por el segmento de red sobre el que ha sido desplegado. A pesar de su indudable valor como elemento de alerta temprana hay varios factores a considerar.

En redes conmutadas: en una red conmutada las comunicaciones se establecen mediante un circuito propietario en lugar de por un canal común como en una red convencional. Para solucionar esto se ha de configurar el NIDS para que escuche el puerto de monitorización de la red.

En redes de alta velocidad: cuando se despliega un NIDS la velocidad de análisis es un factor muy importante a tener en cuenta. Un IDS sin la suficiente velocidad de procesamiento no analizará todo el tráfico de la red. Esto puede ser usado por un atacante, el cual sobrecargará la red para poder introducir exploits que no serán detectados. Para solucionar esto los diferentes proveedores (Cisco, Palo Alto...) ofrecen soluciones con hardware dedicado para mejorar el rendimiento. Otra posible solución es disminuir la velocidad de la red para adecuarla a la velocidad de proceso del IDS.

Redes codificadas: si el atacante hace uso de una conexión SSH el NIDS no será capaz de detectar el ataque. En este caso un HIDS deberá de ser emplazado en la máquina que ofrece la conexión SSH para vigilar su comportamiento.

Si por alguna razón se ignoran estos factores la efectividad del NIDS quedará comprometida hasta el punto de poder resultar inútil.

2.3.3. Clasificación por estrategia de detección

La estrategia de análisis se refiere a la aproximación elegida para distinguir el uso legítimo y el uso anómalo del sistema. La mayor parte de autores coinciden al identificar dos categorías principales y una derivada de ellas: IDS basados en detección de firmas, IDS basados en detección de anomalías y por último IDS híbridos. En pocas palabras la detección de firmas construye modelos sobre ataques conocidos llamados firmas, cualquier coincidencia de una secuencia de eventos con un patrón almacenado disparará una alarma, por otro lado en la detección de anomalías intenta modelar el comportamiento usual del entorno monitorizado y se asume que cualquier evento anormal es en si mismo sospechoso.

IDS basados en detección de firmas

Los IDS que basan su estrategia de detección de firmas monitorizan en el uso indebido de los recursos y realizan su trabajo mediante la clasificación de los eventos de acuerdo a una base de conocimiento que guarda firmas de ataques conocidos. Si una secuencia de eventos de sistema coincide con una firma almacenada en la base de datos se disparará una alarma.

Originalmente las firmas eran construidas a mano usando en conocimiento de expertos, sin embargo hoy en día este proceso se ha agilizado gracias a la utilización de técnicas de minería de datos sobre registros de intrusiones conocidas [27] , [21].

Estos sistemas funcionan de manera muy precisa siempre que se trate de patrones de ataques conocidos, sin embargo resulta bastante fácil engañarlos con la introducción de variaciones equivalentes a nops en los eventos de los ataques, y son básicamente inútiles frente a ataques de nueva aparición [**?**] .

IDS basados en detección de anomalías

Por el contrario los orientados a la detección de anomalías, se basan en el modelado de la actividad normal del sistema y en la detección de desviaciones sobre la misma. Han recibido una gran atención en los últimos años debido a que tienen la interesante propiedad de poder hacer frente a amenazas de nueva aparición. Sin embargo presentan varios problemas relacionados tanto con su proceso de construcción como con el elevado ratio de falsos positivos que suelen arrojar.

El primer obstáculo está en establecer cuál es el uso normal del sistema. Esto no es trivial, y todo IDS basado en anomalías ha de, o bien construir un conjunto de datos limpio, u obtenerlo de algún modo. Decimos limpio puesto que aunque recientemente estos IDS pueden tolerar cierto ruido o presencia de ataques en el conjunto de entrenamiento. Si se entrenan con tráfico real este puede contener ataques no detectados y el detector puede aprender esos ataques como parte del comportamiento normal del sistema, y por lo tanto serán incapaces de detectarlo una vez desplegados.

Por otro lado los conjuntos construidos artificialmente pueden ser demasiado restrictivos y no reflejar en absoluto el comportamiento del sistema, haciendo de esta manera que el número de falsos positivos se dispare. El elevado ratio de falsos positivos, es el precio a pagar por tener un detector de anomalías, puesto que cualquier nueva aplicación o incluso evolución en el uso del mismo por parte de los usuarios puede ser identificado como una amenaza y disparar una alerta.

Cabe reseñar que ante amenazas conocidas, los IDS basados en uso indebido presentan un mejor rendimiento y un coste menor. Sin embargo hay que ponderar la detección precisa de amenazas conocidas frente a la detección de amenazas de nueva aparición [1] .

IDS híbridos

Existe también la posibilidad de combinar ambas estrategias dando lugar a detectores híbridos, los cuales intentan combinar los puntos fuertes de las estrategias antes descritas puesto que los detectores de anomalías no constituyen una respuesta completa al problema de la detección de intrusiones y usualmente han de ser combinados con estrategias de detección de uso indebido.

Las razones para esto son: alto índice de aciertos y bajo número de falsos positivos ante vulnerabilidades conocidas de los IDS basados en uso indebido, y la protección frente a nuevos ataques de los IDS basados en la de detección de anomalías. Un ejemplo de este tipo de IDS es Emerald [23], que realiza ambos tipos de análisis. Como otra característica interesante de Emerald es que hace uso de la plataforma CIDF para permitir la comunicación entre sus elementos de análisis y la adición de módulos procedentes de terceras partes permitiendo así configurar el sistema de detección para responder a las necesidades de la organización que lo despliegue.

2.3.4. Técnicas de análisis

También se puede realizar la clasificación orientandonos por la técnica elegida para implementar el clasificador de eventos en los módulos de análisis del IDS. Clasificamos los métodos en 5 grupos: basados en conocimiento, basados en análisis de firmas, análisis de transición de estados, basados en métodos estadísticos y por último aquellos basados en aperdizaje automático.

Sistemas basados en el conocimiento:
Basados en reglas de tipo if-then-else , si algún evento captado del entorno satisface todas las precondiciones de una regla, esta se disparará. Ofrecen la ventaja de separar la lógica de control del dominio del problema, sin embargo no son una buena alternativa para analizar secuencias de eventos. Ejemplo de este tipo de IDS es P-Best, aunque este sea más bien un intérprete utilizado por otros IDS como [22] , [23] .

Análisis de firmas:
Observa la ocurrencia de cadenas especiales en los eventos monitorizados por el IDS y los compara con los almacenados en una base de datos con firmas de ataques. El principal problema con este enfoque es la necesidad de incorporar una nueva firma a la base de datos por cada nuevo ataque conocido. Quizás el ejemplo más conocido de estos IDS sea Snort [24] debido a su licencia libre y capacidad de desarrollo de preprocesadores y reglas por parte de la comunidad.

Análisis de transición de estados:
Se trata de autómatas finitos que a través de transiciones basadas en los estados de seguridad del sistema intentan modelar el proceso de una intrusión. Cuando el autómata alcanza un estado considerado de peligro lanza una alerta. Representantes de estos sistemas que alcanzaron gran difusión son NetSTAT y USTAT (basados en red y local respectivamente) [25] .

Sistemas basados en métodos estadísticos:
Se basan en modelizar el comportamiento de los usuarios en base a métricas extraídas (estrategia de detección de anomalías) de los registros auditorías del sistema. Una característica clásica de estos sistemas es el establecimiento de perfiles para los distintos usuarios del sistema [26] .

El proceso de comparación consiste en cotejar un vector de variables que han sido recogidas del sistema, con otro almacenado que guarda el comportamiento normal de ese usuario. Si la desviación supera un valor predefinido, se lanzará una alarma. Es conveniente actualizar los perfiles cada cierto tiempo, usando para ello los registros de actividad, del usuario, más recientes, para de esa manera moderar el número de falsos positivos de la herramienta [4]. Sin embargo esta es una fuente de vulnerabilidad de estos sistemas, puesto que un atacante puede desentrenar la herramienta para que permita una intrusión. Un ejemplo de estos sistemas es el ISA-IDS [25].

Sistemas basados en aprendizaje automático:

Se trata de sistemas que realizan uso de técnicas que no necesitan intervención humana para extraer los modelos que representan bien ataques, o bien el uso legítimo del sistema. La extracción de los modelos se realiza durante la fase de entrenamiento [23]. Ejemplos de conjuntos representativos disponibles en la red son los elaborados por la DARPA en 1998

y 1999 así como el proveniente de la KDD Cup de 1999 organizada por la ACM. También pueden ser entrenados con datos recogidos de redes reales. Con respecto al mecanismo de entrenamiento se puede hacer la división entre entrenamiento supervisado y no supervisado.

El aprendizaje supervisado hace uso de conjuntos etiquetados, teniendo como desventaja la escasez de dichos conjuntos de datos sobre sistemas reales, a proteger por la herramienta, ya que pocas organizaciones guardan conjuntos de datos exhaustivamente etiquetados.

Por el contrario el aprendizaje no supervisado no precisa de conjuntos etiquetados. Muchas técnicas de aprendizaje automático pueden hacer uso de ambos métodos en distintas etapas del entrenamiento. Por ejemplo [28] hace uso de ambas.

Estos sistemas han recibido gran atención tanto en el desarrollo de IDS orientados a detección de anomalías como en el de uso indebido. Ya que permiten la extracción de conocimiento sobre el dominio del problema, minimizando la intervención humana y el tiempo que se tarda en preparar las defensas frente a nuevas amenazas.

2.4. Proceso de intrusión

Los IDS han evolucionado acorde a las distintas estrategias desarrolladas por los atacantes. Diversas taxonomías han sido propuestas para la clasificación de los ataques:

- **Bishop** presenta una taxonomía para clasificar los ataques contra el sistema operativo Unix basada en ejes [10] .
- **Howard** propone una taxonomía orientada al proceso del ataque [11] .
- **Lough** desarrolla una taxonomía basada en las vulnerabilidades explotadas por el ataque [12], fue usada con éxito para detectar fallos en el protocolo IEEE 802.11.

Sin embargo no es el propósito de este documento establecer una clasificación exhaustiva de los distintos ataques por lo que la siguiente clasificación solo pretende ilustrar las distintas fases de una intrusión. Nótese que el orden y realización de las distintas fases por el atacante variarán según los objetivos.

Fase 1 (Decisión): El atacante decidirá el tipo de ataque a emprender. Puede tratarse de un ataque de denegación de servicio (DoS), escala de privilegios para acceder a recursos más protegidos del sistema (ej: troyano, registro de teclado, acceder desde un usuario con más privilegios podría evitar al IDS) o bien simplemente un acceso no autorizado a un servicio (ej: exploits, inyección de código SQL).

Fase 2 (Reconocimiento): El objetivo es el de conocer la mayor información del blanco del ataque, desde el sistema operativo y sus servicios de red, hasta datos personales o por ejemplo frecuencias de acceso al sistema (ingeniería social). Los métodos usados durante esta etapa son variados (ej: whois, ICMP Ping, reconocimiento de puertos) y no todos ellos son detectables por las herramientas de seguridad, y si lo son, pueden

enmascararse como usos legítimos del sistema.

Fase 3 (Ataque): Se trata de explotar una debilidad conocida en algún componente del sistema, ya sean errores de programación en aplicaciones, fallos en la gestión de recursos por parte del sistema operativo, o protocolos de comunicación vulnerables. Normalmente el primer paso es inhabilitar el IDS y resto de defensas del sistema. Esto puede lograrse de distintas maneras desentrenar un IDS basado en anomalías, o denegar el servicio a la estación de trabajo que lo ejecuta. A continuación se ejecuta el ataque. Estos se pueden dividir esencialmente en 3 categorías:

1. Ataques de denegación de servicio (ej: ataque al servicio FTP de Microsoft, ataques de denegación de recursos).

2. Ataques mediante exploits remotos (desbordamiento de *buffer*, inyección SQL).

3. Suplantación de identidad (captación de contraseñas en accesos remotos desde redes no seguras, cross-site scripting , spoofing).

Fase 4 (Borrado de huellas): Todo ataque bien planeado incluye esta fase, se centra en borrar todo registro de la intrusión en el sistema así como en el camino recorrido para atacarlo.

El uso creciente de sistemas informáticos en contextos expuestos, como pueden ser redes públicas inalámbricas, junto con el crecimiento exponencial de pequeñas redes ha incrementado el número de vulnerabilidades a explotar por un posible atacante [13] . Si a este fenómeno añadimos que estas redes tienen una pobre o incluso nula gestión de la seguridad, muchas de ellas han sido usadas para llevar a cabo ataques contra objetivos mayores y en principio bien protegidos [14] .

2.5. Técnicas complementarias

En este apartado presentaremos varias técnicas que a pesar de no pertenecer al específicamente al dominio de los sistemas de detección de intrusiones mejoran el rendimiento de estos. Trataremos con aproximaciones a la correlación de alertas cuyo objetivo es reducir el número de falsos positivos, minería de datos para extraer conocimiento de manera más rápida sobre los ataques y el uso de tarjetas aceleradoras de gráficos para mejorar la velocidad de proceso de IDS basados en software.

2.5.1. Correlación de alertas

Similitud de alertas:

Las técnicas basadas en similitud de alertas compararán con todas las alertas y amenazas que tengan atributos o características parecidas (ej: dirección IP origen, IP destino, puertos objetivo). Las alertas cuyos atributos posean un alto nivel de similitud y las agrupa en alertas generales ya existentes o crea nuevas si ninguna de las existente encaja dentro del intervalo de confianza definido. En esta aproximación se trabaja en tres fases: primera fase de agrupamiento, segunda fase comprobación de que múltiples sensores detectan la

misma amenaza y fase tres fusionar las alertas para proporcionar una visión de conjunto del ataque [34]. En la Figura 2.5 vemos el equema de funcionamiento de esta técnica.

Figura 2.5: Esquema similitud de alertas

Escenarios predefinidos de ataque:

En esta aproximación se tiene en cuenta que un ataque por norma general requiere de varias acciones para tener éxito. Todo escenario de ataque contemplado está constituido por una serie de pasos que han de ser efectuados para llevarlo a cabo, por lo tanto las alertas de bajo nivel originadas en los IDS son agrupadas y comparadas con los eventos generados por cualquiera de los escenarios contemplados. Esta técnica está restringida a ataques conocidos y se encuadra dentro de la detección mediante comparación de firmas. En el origen de está aproximación se encuentra el proyecto MIRADOR del DGA/CASSI de Frédéric Cuppens [32]. Cabe reseñar también que este método es coherente con la taxonomía sobre ataques informáticos elaborada por Howard en la que se contempla todo el proceso de intrusión, desde la motivación hasta la consecución del ataque y no solo los medios para llevarlo a cabo [11] . La figura 2.5 ilustra esta técnica.

Pre-requisitos y consecuencias del ataque:

Esta técnica trabaja a un nivel más alto que la correlación por similitud pero más bajo que los sistemas enfocados a escenarios de ataque. Las pre-condiciones se definen como aquellas circunstancias que se deben de dar para que un ataque tenga éxito, las consecuencias por su parte es el estado del sistema después de que un ataque en concreto haya ocurrido. Esta técnica al contrario que en los escenarios predefinidos no está restringida a ataques conocidos ya que descubre la relación causal entre las alertas y no su pertenencia a una secuencia conocida. Sin embargo de las tres técnicas expuestas es la que mayor número de falsos positivos genera [33]. En la figura 2.6 se pueden apreciar los distintos pasos de está tecnica y como se relacionan entre si.

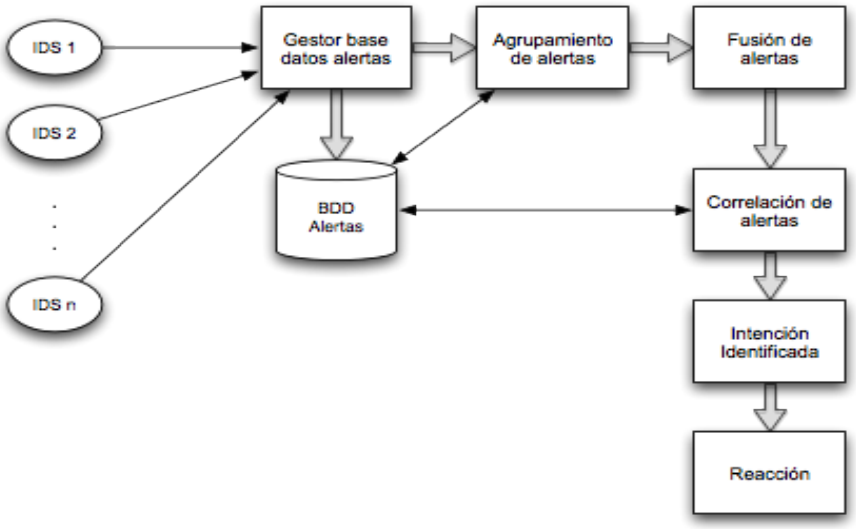

Figura 2.6: Esquema escenarios predefinidos de ataque

Análisis estadístico causal:

En esta técnica se implementa detección por anomalías y usa el test de causalidad de Granger (análisis de series temporales) para relacionar eventos haciendo énfasis en el escenario de ataque. Utiliza la agrupación de alertas para combinar las alertas en bruto generadas por los detectores individuales en alertas de mayor nivel.

Posteriormente se priorizan las alertas y se conduce el análisis del escenario de ataque para correlacionar las alertas con un escenario de ataque. Está anáisis es enteramente estadístico y no tiene necesidad de modelos predefinidos sobre el escenario de ataque por lo tanto nuevos escenarios de ataque pueden ser identificados. En la figura 2.7 podemos apreciar lo anteriormente explicado

Cabe destacar que esta aproximación no es una solución integral al problema de la correlación de alertas pero puede ser usado como parte de un sistema más grande para descubrir nuevos escenarios sobre datos auditados y proveer de modelos para la generación de meta-alertas [35].

Formato de alertas intercambiadas

Como requisito para que estas técnicas puedan ser aplicadas es necesario que todos los detectores que estén trabajando sobre un mismo sistema compartan un formato común para las alertas y que este represente adecuadamente las características tanto del detector que lanzó la alerta (ej: tipo de entorno monitorizado por ese detector en cuestión, técnica empleada) como información especifica de la alerta (ej: código de tiempo, vulnerabilidad afectada). Un ejemplo de formato para la correcta representación de las alertas es el definido por el IDWG que hemos tratado en este mismo capítulo.

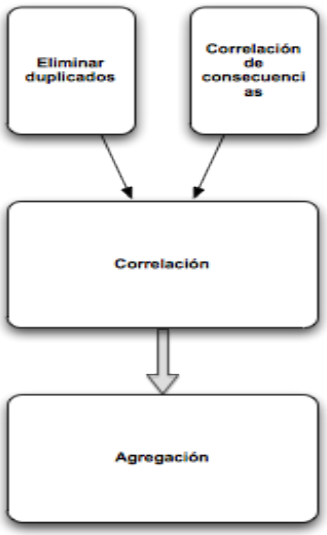

Figura 2.7: Esquema pre-requsitos y consecuencias del ataque

2.5.2. Minería de datos

La minería de datos es un conjunto de técnicas de aprendizaje automático, como tal puede ser incluido dentro del punto correspondiente dentro de este documento. Sin embargo debido a su relevancia que ha adquirido desde que W.Lee y S.Stolfo en 1998 comenzaran a aplicar técnicas de minería de datos en la construcción de modelos para la detección de intrusiones [Lee98] es tratada dentro de este punto[36] .

La construcción de modelos es un punto crucial en la construcción de cualquier IDS ya sea mediante detección de firmas o de anomalías el modelar en el primer caso una intrusión y en el segundo el comportamiento normal del sistema no es trivial, tradicionalmente esta tarea se confiaba a expertos. sin embargo este es un proceso lento, propenso a errores y meramente reactivo. Un experto humano difícilmente descubrirá ataques en un registro de auditoría permaneciendo estos ocultos y el sistema vulnerable, por el otro lado pedir a un administrador que modele el comportamiento normal de una red con multitud de equipos cada uno con su propia configuración es algo parecido al tormento de Sísifo para cuando haya acabado tendrá que volver a empezar ya que para ese momento el modelo extraído será inservible (imaginemos una red extensa con mas de medio centenar de equipos).

La detección de intrusiones basada en minería de datos proporciona la oportunidad de aprender un modelo generalizado. Este modelo puede detectar ataques hasta ese momento nunca vistos mediante el análisis de datos en bruto procedentes de BSM de sistemas UNIX o de herramientas ampliamente difundidas como *Network Fligth Recorder*. Así mismo es capaz de proporcionar modelos que son sensibles al coste de las intrusiones y eficientes en tiempo de computación a la hora de ejecutar la comparación con los datos de entrada [21] . Así mismo las técnicas de correlación de alertas hacen uso extensivo de la minería de datos para conseguir agrupar la información proveniente de cada uno de los sensores individuales [21] [33] .

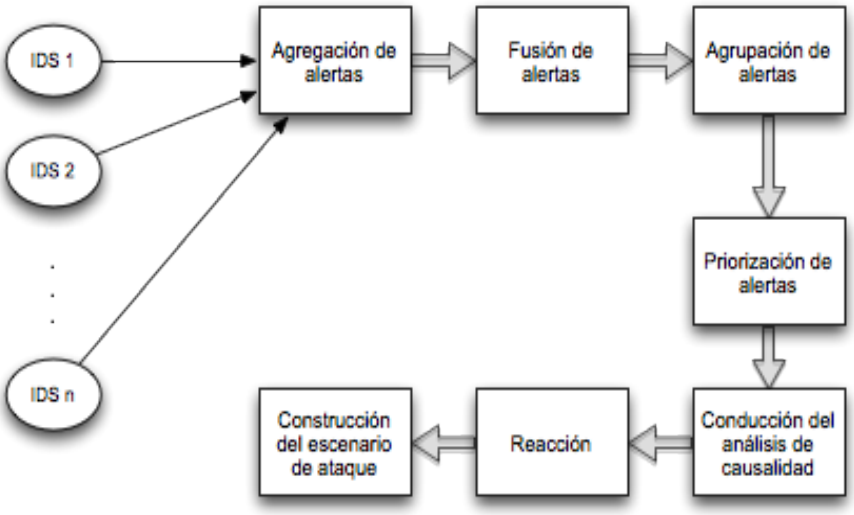

Figura 2.8: Esquema análisis estadístico causal

2.5.3. Computación gráfica de propósito general

La Computación gráfica de propósito general o en inglés *General-Purpose Computing on Graphic Processor Unit* GPGPU es un concepto reciente que trata de hacer uso de la potencia de computo, gran paralelismo y bajo coste de los chips de aceleración gráfica actuales para fuera del ámbito de los gráficos de ordenador [36]
.

Ejecutar un IDS resulta bastante costoso en términos de los recursos que consume para realizar su análisis, especialmente este problema afecta a los NIDS ya que realizan un análisis sobre información de más bajo nivel. Esto incluso en NIDS que no realicen un análisis in-line de los datos constituye una fuente de problemas, por ejemplo un NIDS que no pueda procesar todo el tráfico entrante terminará por llenar su *buffer* de entrada y comenzará a descartar paquetes. Esto es usado intencionalmente por los atacantes como medio para deshabilitar el NIDS y abrir agujeros en las defensas del sistema.

Las soluciones usadas para paliar este tipo de problemas han sido tradicionalmente bien reducir la velocidad de la red para ajustarse a la velocidad de procesamiento del NIDS, bien implementar los algoritmos de detección sobre hardware reconfigurable. En el primer caso obtenemos una mayor seguridad a costa degradación de los servicios ofrecidos por nuestra red, en el segundo obtenemos una mayor seguridad pero perdemos flexibilidad y típicamente exige acudir a soluciones costosas en términos económicos usualmente un NIDS hardware de un proveedor conocido (ej: Cisco) viene a costar varios cientos de dólares tan solo para las versiones básicas de hasta 80 mbps de proceso (Según fabricante).

El uso de la capacidad de procesamiento del flujo de datos de las tarjetas acleradoras gráficas con la que actualmente cuentan la gran mayoría de los ordenadores personales supone una solución al problema anterior sin la necesidad de disponer de hardware dedicado.

Capítulo 3

Snort NIDS de código libre

Snort es un IDS o Sistema de detección de intrusiones basado en red (NIDS). Implementa un motor de detección de ataques y barrido de puertos que permite registrar, alertar y responder ante cualquier anomalía previamente definida. Entre estas anomalías encontramos: firmas de ataques, barridos, intentos de explotar vulnerabilidades, análisis de protocolos conocidos. Esto puede ser llevado a cabo también en modo in-line.

Este IDS implementa un lenguaje de creación de reglas flexibles, potente y sencillo. Durante su instalación ya nos provee de cientos de reglas para la detección de puertas traseras, DDOS, finger, FTP, ataques web, CGI, reconocimientos Nmap, etc. Adicionalmente Snort se puede configurar para que se adapte a la red protegida, permite crear reglas propias, o realizar modificaciones de las existentes.

Puede funcionar como sniffer (podemos ver en consola y en tiempo real qué ocurre en nuestra red, todo nuestro tráfico), registro de paquetes (permite guardar en un archivo los logs para su posterior análisis, un análisis fuera de linea) o como un IDS normal (en este caso NIDS).

Snort (http://www.snort.org/) está disponible bajo licencia GPL. Funciona bajo plataformas Windows, GNU Linux y OSX. Se trata de una de las alternativas más usadas, disponiendo de una gran cantidad de filtros o patrones ya predefinidos, así como actualizaciones frecuentes ante casos de ataques, barridos o vulnerabilidades publicadas en los distintos boletines de seguridad y por parte de su extensa comunidad de usuarios.

3.1. Estructura de Snort

Snort es una herramienta madura con un largo recorrido en el campo de la seguridad informática y como tal a nivel estructural presenta cierta complejidad que a continuación explicaremos. La arquitectura de Snort se ajusta al modelo genérico de arquitectura de IDS basado en reglas:

Después de la fuente de datos Snort incluye un decodificador de paquetes para facilitar su análisis posterior por el resto de los módulos. Para mejorar el análisis de los paquetes Snort incluye preprocesadores que pueden activarse o desactivarse según nuestras necesidades, mejorando los resultados finales obtenidos en la detección de ataques. En Snort los módulos de comparación y de detección se fusionan en un solo módulo, el motor de detec-

ción. Dicho motor puede recibir soporte de otros detectores auxiliares para la detección de ataques específicos, los plug-in de detección.

3.1.1. Elementos de la arquitectura de Snort

- **Packet Capture Library:** Se encarga de la captura de paquetes. Para ello usa las librerías libpcap (en Linux) y WinPcap (en Windows).
- **Packet Decorer:**Toma los paquetes y los decodifica. Primero la trama de la capa de enlace, después el protocolo IP y, por último los paquetes TCP o UDP. Cuando termina la descodificación, Snort tiene toda la información de los protocolos en los lugares adecuados para un posterior análisis.
- **Preprocesador:** El preprocesamiento permite a Snort analizar posteriormente los paquetes más fácilmente. Puede generar alertas, clasificar los paquetes o filtrarlos.
- **Detection Engine:** Analiza los paquetes que le llegan de acuerdo a las reglas que tiene especificadas.
- **Detection (plug-ins):** Módulos de detección auxiliares al motor de detección para análisis más específicos.
- **Output:** Cuando salta una alarma Snort tiene diversos métodos de salida logins, bases de datos y logs de sistema.

3.1.2. Recorrido de los paquetes de Snort

En este apartado veremos el camino que recorren los datos dentro de la arquitectura de Snort:

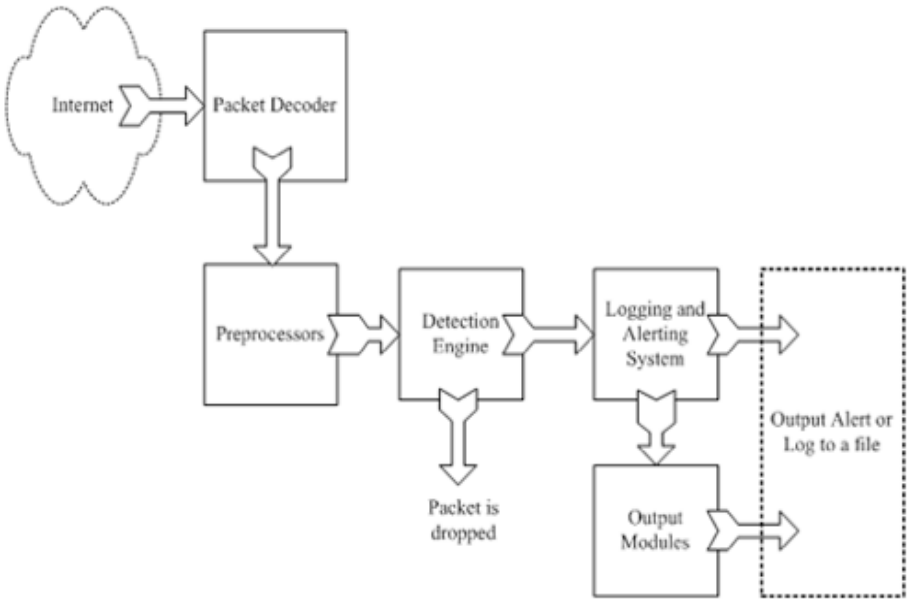

Figura 3.1: Recorrido de los paquetes capturados en snort

1. Se capturan los paquetes provenientes de la red (librería de captura de paquetes) y se envían al descodificador de paquetes.

2. Los paquetes se descodifican basándose en el protocolo al que correspondan (descodificador de paquetes). Una vez descodificados, los paquetes son enviados a los preprocesadores.

3. Los paquetes son clasificados y filtrados. También es posible que ya se generen algunas alarmas (por parte de los preprocesadores). Las alarmas generadas serán notificadas al **módulo salida** (y se le adjuntará el paquete que generó dicha alarma) y los paquetes ya clasificados y filtrados se envían al motor de detección.

4. Se analizan los paquetes clasificados y filtrados en el motor de detección. Dicho análisis se realiza mediante la búsqueda de correspondencias entre dichos paquetes y el juego de reglas con el que trabaja Snort (para saber más acerca de las reglas ver el apartado "Reglas"). A su vez el motor de detección recibirá el soporte de los **plug-ins de detección** en caso de que sea necesario. Si este análisis hiciese saltar alguna alerta se notificaría al **módulo salida** y se le adjuntaría el paquete que la generó. En caso contrario se permite al paquete el paso al sistema.

5. Una vez que llega el paquete que generó la alarma al **módulo salida**, este se encarga de actualizar el fichero **Alerts.ids** (en el que se almacena qué paquete generó una alerta y por qué) y el fichero Snort.log.xxxxx (en el que se almacena el contenido de dicho paquete de una forma más o menos detallada en función del formato de las alertas en el que se estuviese ejecutando Snort).

3.2. Modos de ejecución Snort

Snort puede configurarse para ejecutarse en 4 modos distintos de ejecución: modo monitor, modo registrador de paquetes, modo Network Intrusion Detection System (NIDS) y modo In-line. A continuación, veremos las características y el funcionamiento de cada uno de ellos.

3.2.1. Modo monitor

Snort puede utilizarse como un monitor del tráfico que pasa por una red. El programa lee los paquetes y los muestra por consola según los va capturando. Podemos por ejemplo:

1. Mostrar por consola cabeceras de paquetes TCP/UDP detectados:

```
snort -vd

Running in packet dump mode
Log directory = log
Initializing Network Interface
\Device\NPF_{604C8AE3-5FAC-45A5-BFAA-81175A8C32BF}
.....
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
05/21-11:06:18.962018 192.168.4.5:3890 -> 192.168.4.15:8080
```

```
TCP TTL:128 TOS:0x0 ID:33217 IpLen:20 DgmLen:681 DF
***AP*** Seq: 0xE3A50016 Ack: 0x8B3C1E4D Win: 0xFAF0 TcpLen: 20
47 45 54 20 68 74 74 70 3A 2F 2F 77 77 77 2E 6F GET http://www.x
6D 65 6C 65 74 65 2E 63 6F 6D 2E 62 72 2F 73 75 xxxx.com.br/xx
70 65 72 6F 6D 65 6C 65 74 65 2F 64 6F 77 6E 6C xxxxxxx/downl
6F 61 64 73 2F 64 65 66 61 75 6C 74 2E 61 73 70 oads/default.asp
20 48 54 54 50 2F 31 2E 30 0D 0A 55 73 65 72 2D HTTP/1.0..User-
41 67 65 6E 74 3A 20 4D 6F 7A 69 6C 6C 61 2F 34 Agent: Mozilla/4
2E 30 20 28 63 6F 6D 70 61 74 69 62 6C 65 3B 20 .0 (compatible;
4D 53 49 45 20 36 2E 30 3B 20 57 69 6E 64 6F 77 MSIE 6.0; Window
73 20 4E 54 20 35 2E 30 29 20 4F 70 65 72 61 20 s NT 5.0) Opera
37 2E 31 31 20 20 5B 65 6E 5D 0D 0A 48 6F 73 74 7.11 [en]..Host
3A 20 77 77 77 2E 6F 6D 65 6C 65 74 65 2E 63 6F : www.xxxxx.co
6D 2E 62 72 0D 0A 41 63 63 65 70 74 3A 20 74 65 m.br..Accept: te
78 74 2F 68 74 6D 6C 2C 20 69 6D 61 67 65 2F 70 xt/html, image/p
6E 67 2C 20 69 6D 61 67 65 2F 6A 70 65 67 2C 20 ng, image/jpeg,
69 6D 61 67 65 2F 67 69 66 2C 20 69 6D 61 67 65 image/gif, image
2F 78 2D 78 62 69 74 6D 61 70 2C 20 2A 2F 2A 3B /x-xbitmap, */*;
71 3D 30 2E 31 0D 0A 41 63 63 65 70 74 2D 4C 61 q=0.1..Accept-La
......
```

2. Podemos también lanzar el comando anterior con la version verbose aceptando también mostrar cabeceras IP e ICMP:

```
snort {vd

Running in packet dump mode
Log directory = log
Initializing Network Interface
\Device\NPF_{604C8AE3-5FAC-45A5-BFAA-81175A8C32BF}

.....

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

05/21-11:06:18.943887 192.168.4.5:3890 -> 192.168.4.15:8080
TCP TTL:128 TOS:0x0 ID:33216 IpLen:20 DgmLen:40 DF
***A**** Seq: 0xE3A50016 Ack: 0x8B3C1E4D Win: 0xFAF0 TcpLen: 20

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

05/21-11:06:18.962018 192.168.4.5:3890 -> 192.168.4.15:8080
TCP TTL:128 TOS:0x0 ID:33217 IpLen:20 DgmLen:681 DF
***AP*** Seq: 0xE3A50016 Ack: 0x8B3C1E4D Win: 0xFAF0 TcpLen: 20
47 45 54 20 68 74 74 70 3A 2F 2F 77 77 77 2E 6F GET http://www.x
6D 65 6C 65 74 65 2E 63 6F 6D 2E 62 72 2F 73 75 xxxx.com.br/xx
70 65 72 6F 6D 65 6C 65 74 65 2F 64 6F 77 6E 6C xxxxxxx/downl
```

```
6F 61 64 73 2F 64 65 66 61 75 6C 74 2E 61 73 70 oads/default.asp
20 48 54 54 50 2F 31 2E 30 0D 0A 55 73 65 72 2D HTTP/1.0..User-
41 67 65 6E 74 3A 20 4D 6F 7A 69 6C 6C 61 2F 34 Agent: Mozilla/4
2E 30 20 28 63 6F 6D 70 61 74 69 62 6C 65 3B 20 .0 (compatible;
4D 53 49 45 20 36 2E 30 3B 20 57 69 6E 64 6F 77 MSIE 6.0; Window
73 20 4E 54 20 35 2E 30 29 20 4F 70 65 72 61 20 s NT 5.0) Opera
37 2E 31 31 20 20 5B 65 6E 5D 0D 0A 48 6F 73 74 7.11 [en]..Host
3A 20 77 77 77 2E 6F 6D 65 6C 65 74 65 2E 63 6F : www.xxxxx.co
6D 2E 62 72 0D 0A 41 63 63 65 70 74 3A 20 74 65 m.br..Accept: te
78 74 2F 68 74 6D 6C 2C 20 69 6D 61 67 65 2F 70 xt/html, image/p
6E 67 2C 20 69 6D 61 67 65 2F 6A 70 65 67 2C 20 ng, image/jpeg,
69 6D 61 67 65 2F 67 69 66 2C 20 69 6D 61 67 65 image/gif, image
2F 78 2D 78 62 69 74 6D 61 70 2C 20 2A 2F 2A 3B /x-xbitmap, */*;
71 3D 30 2E 31 0D 0A 41 63 63 65 70 74 2D 4C 61 q=0.1..Accept-La
......
```

3.2.2. Modo registro de paquetes

Este modo de ejecución es similar al modo monitor, con la diferencia de que en lugar de mostrar los paquetes capturados por consola, estos se almacenan en disco

```
snort -dev -l ./log
```

-l indicará a Snort que quiere guardar el informe en un directorio determinado generando allí los registros y estructuras de información necesarias. Con -h indicaremos la dirección IP a registrar y con –b le diremos que genere los registros binarios, Este tipo de logs pueden ser tratados por herramientas como TCPDump o Windump.

```
snort -vde -l ./log -h 192.168.4.0/24
Running in packet logging mode
Log directory = ./log

snort -l ./log -b
Running in packet logging mode
Log directory = ./log
```

Si usamos –b no hace falta indicarle el –h, -de o –v ya que guardará todos los datos del tráfico. Con –i le podremos indicar la interface a usar en la captura:

```
snort -vde -i 1
# snort -vde -i eth0
snort -W
```

Con –W vemos las interfaces disponibles. En caso de necesidad podemos ejecutar snort sin opciones para poder ver ver todos los posibles parámetros. Si queremos mostrar por pantalla alguno de estos paquetes que hemos registrado (siendo packet.log el nombre del paquete que queremos mostrar por consola) usamos el comando:

```
./snort -dv -r packet.log
```

3.2.3. Filtros

Podemos añadir, a parte de las opciones, una serie de filtros para optimizar los resultados obtenidos. Estos filtros se añadirán en el mismo formato que usa programas como TCPDump ya que usan las mismas librerías de captura de paquetes (libpcap). Ejemplo:

```
snort -vd host 192.168.4.5 and dst port 8080

Running in packet dump mode
Log directory = ./log

Initializing Network Interface eth0

--== Initializing Snort ==--
Initializing Output Plugins!
Decoding Ethernet on interface eth0
--== Initialization Complete ==--

-*> Snort! <*-

...

07/15-12:34:26.059644 192.168.4.5:4533 -> 192.168.4.15:8080
TCP TTL:128 TOS:0x0 ID:16544 IpLen:20 DgmLen:40 DF
***A**** Seq: 0xC47AC53E Ack: 0xC83C2362 Win: 0xFAF0 TcpLen: 20

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

07/15-12:34:26.059880 192.168.4.5:4533 -> 192.168.4.15:8080
TCP TTL:128 TOS:0x0 ID:16545 IpLen:20 DgmLen:40 DF
***A**** Seq: 0xC47AC53E Ack: 0xC83C31E4 Win: 0xFAF0 TcpLen: 20

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

Snort aceptará también filtros más avanzados en su modo sniffer:

snort -vd icmp[0] = 8 and dst host 192.168.4.1

snort -vd icmp[0] != 8 and icmp[0] != 0

snort -vd 'udp[0:2] < 1024' and host 192.168.4.1
```

3.2.4. Modo NIDS

Este modo captura el tráfico de la red y lo analiza para detectar intrusiones. Para ello compara el tráfico con una serie de reglas y registra los paquetes que hagan saltar alguna alarma, es decir, que correspondan con algún comportamiento sospechoso previamente definido. Es el modo que vamos a usar en nuestro sistema.

snort -dev -l ./log -h 192.168.4.0/24 -c ../etc/snort.conf

donde .log es la carpeta especificada para registrar los paquetes, 192.168.4.0/24 la dirección de la red a proteger y snort.conf es el fichero de configuración de snort, que como veremos, incluye información sobre reglas y preprocesadores a usar.

Con al opción -D indicará a snort que corra como un servicio. Esto es sólo válido para las versiones : GNU Linux / OSX

```
./snort {A fast -dev -l ./log -h 192.168.1.0/24 -c snort.conf
```

Por defecto, cuando ejecutamos el modo NIDS, primero se aplican las reglas Alert, después las reglas Pass y por último las reglas Log. Modos de alertas:

1. Rápido: El modo Alerta Rápida nos devolverá información sobre: tiempo, mensaje de la alerta, clasificación, prioridad de la alerta, IP y puerto de origen y destino.

   ```
   snort -A fast -dev -l ./log -h 192.168.4.0/24 -c ../etc/snort.conf

   09/19-19:06:37.421286 [**] [1:620:2] SCAN Proxy (8080) attempt [**]
   [Classification: Attempted Information Leak] [Priority: 2] ...
   ... {TCP} 192.168.4.3:1382 -> 192.168.4.15:8080
   ```

2. Completo: El modo de Alerta Completa nos devolverá información sobre: tiempo, mensaje de la alerta, clasificación, prioridad de la alerta, IP y puerto de origen/destino e información completa de las cabeceras de los paquetes registrados.

   ```
   snort -A full -dev -l ./log -h 192.168.4.0/24 -c ../etc/snort.conf

   [**] [1:620:2] SCAN Proxy (8080) attempt [**]
   [Classification: Attempted Information Leak] [Priority: 2]
   09/19-14:53:38.481065 192.168.4.3:3159 -> 192.168.4.15:8080
   TCP TTL:128 TOS:0x0 ID:39918 IpLen:20 DgmLen:48 DF
   ******S* Seq: 0xE87CBBAD Ack: 0x0 Win: 0x4000 TcpLen: 28
   TCP Options (4) => MSS: 1456 NOP NOP SackOK
   ```

3. Socket: Manda las alertas a través de un socket, para que las escuche otra aplicación. Está opción es para Linux/UNIX.

   ```
   snort -A unsock -c snort.conf
   ```

4. Consola: Imprime las alarmas en pantalla.

   ```
   snort -A console -dev -l ./log -h 192.168.4.0/24 -c ../etc/snort.conf
   ```

5. Mudo: Desactiva las alarmas.

   ```
   snort -A none -c snort.conf
   ```

6. SMB: Permite a Snort realizar llamadas al cliente de SMB (cliente de Samba, en Linux), y enviar mensajes de alerta a hosts Windows (WinPopUp). Para activar este modo de alerta, se debe compilar Snort con el conmutador de habilitar alertas SMB (enable -smbalerts). Evidentemente este modo es para sistemas Linux/UNIX. Para usar esta característica enviando un WinPopUp a un sistema Windows, añadiremos a la línea de comandos

```
snort: -M WORKSTATIONS.
```

7. Registro de sistema: Envía las alarmas al registro del sistema

```
snort -A console -dev -l ./log -h 192.168.4.0/24 -c \
../etc/snort.conf -s 192.168.4.33:514
```

3.2.5. Modo *In-line*

El Modo in-line va más allá de las funciones del IDS, haciendo que snort se comporte como un IPS, es decir, un sistema de prevención de intrusos capaz de tomar sus propias decisiones basándose en tablas IP.

En este modo de ejecución de Snort, se obtienen los paquetes de los Iptables (reglas sobre los diferentes paquetes de una red usadas por el administrador del sistema) y se usan nuevas reglas para determinar si se permite o no el paso de esos paquetes. Es decir, los Iptables utilizarán las reglas de Snort para determinar el paso de los paquetes.

Existen tres tipos de acciones sobre los paquetes en este modo:

1. **Descartar:** Los paquetes se dejan pasar o no y se registran mediante los mecanismos usuales de Snort.
2. **Rechazar:** Se realizan las mismas acciones que con las reglas Drop y además se envía un TCP reset si el protocolo es el TCP o un ICMP port unreachable si el protocolo es el UDP.
3. **Rechazo selectivo:** Se deja pasar o no a los paquetes, pero no se registra ninguno.

El Inline Mode es un modo de ejecución ligeramente diferente a los vistos hasta ahora (en un principio era una herramienta adicional que usaba como base Snort). Debido a esto, para poder ejecutarlo la instalación de Snort debe hacerse de la siguiente manera:

```
./configure --enable-inline
make
make install
```

Debemos asegurar que el módulo `ip_queue` está cargado en nuestro sistema. Para ello podemos utilizar el siguiente comando:

```
iptables -A OUTPUT -p tcp --dport 80 -j QUEUE
```

Ahora ya podríamos utilizar Snort en modo inline mediante el siguiente comando:

```
snort_inline -QDc ../etc/drop.conf -l /var/log/snort
```

Capítulo 4

Sistema de Detección de Anomalías de Red basado en el procesamiento de Payload

El proyecto SIstema de Detección de Anomalías de Red basado en el procesamiento de Payload, de ahora en adelante SDARP, es un preprocesador del conocido IDS Snort. Se trata de un detector orientado a la detección de anomalías en el tráfico de red. SDARP utiliza aprendizaje automático para la extracción del modelo de comportamiento normal de la red, con ello se consigue agilizar el proceso de adaptación a nuevos sistemas. Esta característica lo hace en extremo interesante ya que Snort como se ha mostrado en apartados anteriores es un NIDS basado en reglas y como tal orientado a la detección de firmas, por lo que la adición de SDARP le añade capacidades de detección en día cero.

En este apartado se presentará información sobre el método de detección empleado, además entraremos en detalle sobre el proceso de entrenamiento parcialmente supervisado que se ha de seguir y el formato estándar empleado para la visualización de las alertas generadas durante la fase de detección. A continuación se procederá a mostrar los resultados obtenidos en la detección de ataques en conjuntos de datos procedentes de tráfico real facilitados por el Centro de Cálculo de la Universidad Complutense de Madrid y originados en su propia red.

Adicionalmente se introducen los trabajos realizados para mejorar el rendimiento del algoritmo detector mediante el uso de la librería OpenMp y arquitecturas de memoria compartida (en nuestro caso las pruebas fueron realizadas sobre un procesador Intel core2 duo). Así mismo se recopilan los resultados obtenidos para las distintas opciones de paralelización que fueron probadas durante el desarrollo del proyecto.

Por último se hará un análisis de posicionamiento de nuestra aplicación en el mercado actual de Sistemas Detectores de Intrusiones teniendo en cuenta los resultados obtenidos en el anterior apartado.

4.1. Técnica de detección

En el ámbito de los NIDS existen numerosas técnicas de detección de código malicioso. La seleccionada para implementar el preprocesador actual fue Anagram, con una variación para hacerlo más eficiente. Esta técnica es en realidad una evolución de otra llamada Payl. A continuación pasamos a explicar las dos técnicas:

4.1.1. Payl

Esta técnica se basa en la creación de patrones para definir lo que se consideraría como tráfico normal de la red. El filtrado del tráfico se realiza según ciertas características que se considera que definen el contenido de las cargas útiles; a saber: el puerto, el tamaño del paquete y la dirección de flujo. Para la creación de los patrones se usa 1-gram. Esto es para evitar un consumo de memoria excesivo.

En la fase de entrenamiento generaremos los patrones que definirán el comportamiento normal de la red. La implementación de dichos patrones se lleva a cabo con dos *arrays* de 256 posiciones para cada una de las características. Uno de los *arrays* guardará las medias de las frecuencias de cada byte, mientras que el otro contendrá las desviaciones típicas.

Para cada nueva muestra se actualizará el patrón, actualizando ambos *arrays*. No es necesario guardar los datos anteriores, ya que tanto la nueva media como la nueva desviación típica se pueden calcular a partir de sus valores anteriores usando las siguientes ecuaciones:

$$\acute{x} = \frac{\acute{x} \times N + x_{N+1}}{N+1} = \acute{x} + \frac{x_{N+1} - \acute{x}}{N+1}$$

Figura 4.1: Cálculo nueva media para Payl

$$Var(X) = E(X - EX)^2 = E(X^2) - (EX)^2$$

Figura 4.2: Cálculo desviación tipica Payl

Según van llegando las muestras y se analizan, si una de ellas sobrepasa cierto umbral respecto al patrón actual, se desecha. Esta característica permite al sistema ser tolerante a pequeños ataques dentro del tráfico de entrenamiento. También se reduce la cantidad de memoria necesaria mediante la fusión de muestras con histogramas parecidos y tamaños de paquete similares. A la hora de decidir que muestras se fusionan, se establece un umbral para decidir si los tamaños de paquetes son considerados similares. Para calcular el umbral de la similitud de la distribución de sus histogramas se usa la distancia de Manhattan.

En la fase de detección, se compara el tráfico de la red con los patrones generados durante el entrenamiento, pudiendo clasificar así el mismo como legítimo o malicioso. Durante esta fase los patrones también se van actualizando, pudiendo dar más importancia a los cambios en el uso de la red, o a mantener los patrones iniciales. Así mismo, en esta

etapa se calcula el número de veces que aparece cada byte dentro del paquete a analizar, y se compara con la media del patrón adecuado. Para ello se usa la distancia reducida de Mahalanobis, que tiene en cuenta la desviación típica. Si esta distancia es mayor que un cierto umbral, se generará un alerta. La fórmula para calcular la distancia de Mahalanobis se muestra en la figura 4.3.

$$d(x, \acute{y}) = \sum_{i=0}^{n-1} \left(\left| x_i - \acute{y}_i \right| / (\acute{\sigma}_i + \alpha) \right)$$

Figura 4.3: Cálculo distancia Mahalanobis

4.1.2. Anagram

Esta técnica es una mejora de la anterior, las principales ventajas respecto a Payl son:

1. Mayor precisión a la hora de detectar ataques, especialmente los *mimicry*.
2. Computacionalmente eficiente debido al uso de funciones de dispersión en los *bloom filters*.
3. Eficiente en uso de memoria debido al uso de *bloom filters*.
4. Correlación rápida entre distintos sistemas debido a la posibilidad de cambiar la información de los *bloom filters* sin comprometer la privacidad de los datos.
5. Creación de firmas de nuevos ataques usando la correlación de distintos sistemas.

Al igual que Payl, esta técnica se basa en el uso de *n-grams*, pero en este caso de tamaño mayor que 1. Esto es posible gracias al uso de la estructura *bloom filter*. El *bloom filter* es una estructura que sirve para almacenar de forma binaria la aparición o no de un determinado número de *n-gram*, pero no da información sobre la frecuencia de aparición del mismo.

El funcionamiento del *bloom filter* es el siguiente: cuando se procesa un *n-gram*, se pasa por las funciones de dispersión, obteniendo un conjunto de posiciones que lo representan dentro del *bloom filter*. A la hora de buscar un elemento, se vuelve a pasar por las funciones de dispersión, de esta forma, si al menos una de las posiciones que le corresponden tiene el valor 0, querrá decir que ese *n-gram* no ha sido detectado previamente. El número de funciones de dispersión necesarias para optimizar el *bloom filter* viene dado por la formula mostrada en la figura 4.4. Donde **m** es el número de bits en el *array*, y **n** es el número de elementos insertados.

$$\frac{m}{n} \times \ln 2$$

Figura 4.4: Cálculo número de funciones de dispersión

Una de las grandes ventajas del *bloom filter* es el ahorro en espacio que supone. Además, como el coste de las operaciones sobre esta estructura es constante, el rendimiento computacional también es muy bueno. Como desventaja, cabe destacar que esta estructura puede causar falsos positivos al buscar la aparición de un *n-gram*, si las posiciones asociadas al mismo ya contienen el valor de aparición para otro elemento insertado anteriormente, pero nunca dará un falso negativo.

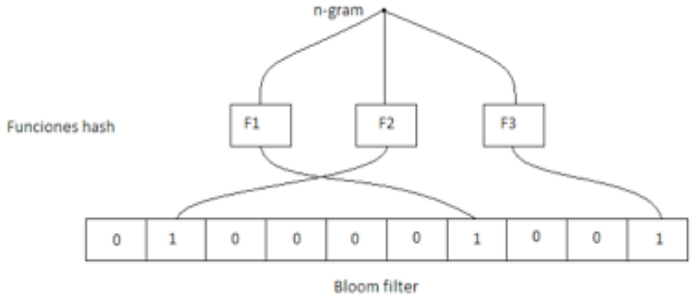

Figura 4.5: Funcionamiento de un *bloom filter*

Durante la fase de entrenamiento, se rellenarán dos *bloom filters*. Uno de ellos contendrá los *n-grams* que representan el tráfico normal de la red, mientras que el otro contendrá *n-grams* que aparecen en ataques conocidos. Este sistema es semi-supervisado, debido a la necesidad de supervisión cuando se usan ataques para rellenar uno de los *bloom filters*.

En la fase de detección se obtienen los *n-grams* que componen cada paquete, y se contrastan con ambos *bloom filters*. A continuación, se le da una puntuación al paquete en función del número de *n-grams* que pertenecen a tráfico normal de la red y de cuantos pertenecen a tráfico sospechoso. Si esta puntuación supera cierto umbral, el paquete se considerará un ataque.

Mediante el uso de *n-grams* de mayor tamaño se tiene en cuenta la dependencia entre bytes, teniendo como consecuencia una mayor precisión en el análisis. Por otro lado, al no tener en cuenta la frecuencia de aparición de los *n-grams*, se pueden detectar ataques ocultos entre tráfico legítimo.

A diferencia de Payl, esta técnica **no es resistente a fallos durante el entrenamiento**, ya que si un paquete malicioso aparece en el tráfico de entrenamiento, el sistema lo considerará como legítimo en el futuro. Para paliar este problema, como ya se ha comentado, se usa un segundo *bloom filter* con ataques conocidos, que se usará durante el entrenamiento del sistema, para determinar si existe algún paquete malicioso en el tráfico de entrenamiento, y de esta forma deshacerse del mismo.

Como última consideración, hay que determinar un tiempo de entrenamiento mínimo que permita al patrón, del tráfico de la red, estabilizarse y así representar de forma correcta el tráfico legítimo de la red. Tampoco conviene superar cierto umbral, ya que el sistema podría sobre entrenarse, aumentando el número de falsos positivos generados en la fase de

detección.

Como complemento a la hora de resistir los ataques *mimicry*, se utiliza una técnica de aleatorización de patrones. Esto se consigue aplicando una máscara binaria aleatoria, que cree fragmentos equilibrados lo suficientemente grandes como para que se puedan extraer *n-grams* de ellos. Todo esto hace muy difícil que el atacante use técnicas de relleno (*padding*) para camuflar el ataque como un patrón de tráfico legítimo.

4.2. Funcionamiento

4.2.1. Etapas de entrenamiento

Se divide en tres etapas.

- **Etapa 1:** En esta etapa se analiza un conjunto de paquetes de tráfico legítimo, que usaremos para rellenar el *bloom filter*. Estos paquetes se analizarán dividiéndolos en *n-grams* de tamaño 3, los cuales se pasarán por las funciones de dispersión para determinar sus posiciones dentro del *bloom filter*. Así mismo, incrementaremos en 1 la posición determinada en el *bloom filter* auxiliar. Una vez terminado este proceso, se utiliza la información almacenada en el *bloom filter* auxiliar para calcular la desviación estándar y rellenar el *bloom filter* binario que se usará en la fase de detección.
- **Etapa 2:** En este momento probamos el detector con tráfico legítimo para calcular la media de *n-grams* sospechosos que se han encontrado en los paquetes. Esta media servirá como cota inferior del umbral de *n-grams* sospechosos que se usará para detectar el código malicioso.
- **Etapa 3:** Esta última etapa consiste en probar el detector con tráfico con ataques para calcular la media de *n-grams* sospechosos que se han encontrado en los paquetes. Esta medida servirá como cota superior del umbral de *n-grams* sospechosos que se usará para detectar el código malicioso.

4.2.2. Pasos del entrenamiento

- Poner en la misma carpeta los ficheros entrenamiento y ProfilerMod (tanto los .c como los .h).
- Compilar ProfilerMod por ejemplo:

```
gcc -o ProfilerMod ProfilerMod.c
```

- Cambiar el fichero snort.conf al modo adecuado según la etapa del entrenamiento que vayamos a ejecutar. Esto se consigue descomentando solo la opción deseada dentro del fichero, manteniendo el resto comentadas.
- Ejecutar el programa con las opciones adecuadas. Hay que asegurarse de que la dirección (si se usa) acabe con el carácter '/':

```
./ProfilerMod -b "dirección de los {\it datasets} a usar" [número]
```

Si el parámetro número no se especifica, el sistema entrenará con los *datasets* en la carpeta hasta que el entrenamiento sea satisfactorio. Si el número es 0 usará todos los archivos dentro de esa carpeta. En otro caso, tomará el número de archivos indicado para entrenar, sea cual sea el resultado del entrenamiento durante dicho proceso (satisfactorio o no).

- Para el entrenamiento de las referencias:

```
./ProfilerMod -r [dirección de los {\it datasets} a usar]
```

Si no se especifica ninguna dirección, usará por defecto las rutas guardadas en un archivo que genera el entrenamiento base. En otro caso usará todos los *datasets* de la dirección proporcionada como parámetro.

- Para el cálculo de las k's:

```
/ProfilerMod -k "dirección de los ataques"
```

Realiza el cálculo de las k's con todos los ficheros de la dirección proporcionada.

- Para la detección de losde los ficheros de la dirección proporcionada:

```
/ProfilerMod -d "dirección de los ataques"
```

Realiza la detección con todos los ficheros de la dirección proporcionada.

4.2.3. Fase de detección

Esta fase procesa los paquetes (sean capturados de una red, o leídos de un archivo) y los compara con los datos almacenados en el *bloom filter* que se ha rellenado en la fase de entrenamiento.

Para esto se dividen los paquetes de la misma manera que en la etapa 1 del entrenamiento, y comprobamos el número de *n-grams* que no aparecen en el *bloom filter*. Si este número es mayor que el umbral máximo, o menor que la cota mínima, el paquete que contiene en *n-gram* será considerado sospechoso y se lanzará una alarma.

4.2.4. Elaboración de los *datasets*

Para el entrenamiento y la comprobación de los resultados de nuestro sistema hemos elaborado un *dataset* (conjunto de pcaps representativos del tráfico y de los ataques conocidos) con la herramienta de captura de tráfico Wireshark cite.

Wireshark es una herramienta que permite ver el tráfico que pasa a través de una red mediante una interfaz gráfica muy intuitiva. Además tiene la ventaja de ser software libre, disponer de infinidad de manuales de uso tanto en inglés como en español y un gran reconocimiento en el mundo de la auditoria de sistemas. Por todo esto decidimos usar esta aplicación para capturar el tráfico que formará parte de nuestros *datasets*.

Para realizar las distintas etapas del entrenamiento del sistema hemos necesitado crear varios conjuntos de datos: unos con tráfico limpio y otros con tráfico con código malicioso. Para la creación de los *datasets* de tráfico limpio se capturó el tráfico de un ordenador doméstico durante varios días. Para poder crear un patrón de comportamiento fiable se realizaban todos los días acciones parecidas y visitas a las mismas páginas web. La acciones capturadas son las siguientes:

- Tráfico generado por aplicaciones de intercambio de archivos P2P.
- Envío de correo electrónico mediante Hotmail con ficheros adjuntos (.doc, .pdf, .mp3, .jpeg...).
- Lectura de correo electrónico en Hotmail con ficheros adjuntos (.doc, .pdf, .mp3, .jpeg...)
- Navegación por páginas web de todo tipo (as, marca, mundo deportivo, diario sport, elpais, el mundo, abc, larazon, fdi ucm, ucm, upm, uam, forocoches, todoexpertos, yahoorespuestas, comunidad de Madrid, gobierno de españa, the washington post, nba, universidad de Berkeley, wikipedia.es, wikipedia.org, cinetube, peliculasyonkis, seriesyonkis, vagos. . .)
- Descarga de programas y de aplicaciones del conocido servidor Softonic.
- Visualización de videos en youtube.

Para la captura del tráfico con ataques teníamos la necesidad de tener conjuntos de datos con un solo ataque aislado en cada uno para poder entrenar y probar bien el sistema. Debido a esto decidimos descargar algunos virus y programas peligrosos conocidos de las siguientes páginas web:

- www.worst-viruses.2ya.com
- www.rigacci.org/comp/virus

Estos programas estaban comprimidos por lo que para poder capturarlos subimos los virus descomprimidos al conocido file hosting MegaUpload y capturamos la descarga de estos creando un archivo pcap por cada virus.

El listado completo de virus capturados es el siguiente: melissa, nimda, loveletter, sasser.b, happy99, netsky.z, blaster, sobig, Bomberman, Bloodlust, DnDdos, MXZ, MXZ II, MMR, OwNeD, Zip Monsta, HDKP4, Die 3, Click 2, Gimp, BitchSlap, NetBus 2.0 Pro, NetDevil 1.5, NYB, Parity.b, WYX.b, Stoned.a, PingPong.a, Form.a, Parity.a, Bye, Junkie.1027, HLLC.Crawen.8306, Macro.Word97.Thus.aa, Macro.Word97.Ethan, Macro.Word.Cap, Macro.Word97.Marker-based, Macro.Word97.Marker.r, Macro.Office.Triplicate.c, Trojan.Win32.VirtualRoot, JS.Fortnight.b, Trojan.PSW.Hooker.24.h, Trojan.JS.Seeker.o, JS.Trojan.Seeker-based, Trojan.Win32.DesktopPuzzle, JS.Trojan.Seeker.b, Backdoor.SdBot.aa, Backdoor.Zenmaster.102, Backdoor.DonaldDick.152, Backdoor.DonaldDick.15, Backdoor.Buttman, Worm.Win32.Fasong.a, Worm.Win32.Lovesan.a, Win32.Xorala, Win32.FunLove.4070, Worm.Win32.Muma.c, Win95.Dupator.1503, Win32.HLLP.Hantaner,

Joke.Win32.Errore, Joke.Win32.Zappa, Worm.Bagle.Z, IIS-Worm.CodeRed.a, I-Worm.Moodown.b, I-Worm.NetSky.d, I-Worm.Rays, I-Worm.Sober.c.dat, I-Worm.LovGate.i, IIS-Worm.CodeRed.c, I-Worm.Sober.c, I-Worm.Mydoom.a, I-Worm.Mimail.a, I-Worm.Tanatos.dam, I-Worm.Swen, Worm.P2P.SdDrop.c, I-Worm.Sobig.f, I-Worm.Klez.e, I-Worm.Sobig.b, I-Worm.Tanatos.b, I-Worm.Fizzer, Worm.Win32.Opasoft.a.pac, I-Worm.Tettona, Worm.Bagle.AG, Trojan.ZipDoubleExt-1, Worm.Mytob.IV, orm.Mytob.BM-2, Worm.SomeFool.Q, Trojan.W32.PWS.Prostor.A, Worm.Mydoom.AS, Worm.Mytob.V.

Este conjunto de virus se puede clasificar en las siguientes categorías:

- Viruses. *Malware* que tiene por objeto alterar el normal funcionamiento de la computadora, sin el permiso o el conocimiento del usuario.
- Worms. *Malware* que tiene la propiedad de duplicarse a sí mismo.
- Nukers. Nombre genérico para ataques de tipo DoS en TCP/IP.
- Trojans. *Malware* capaz de alojarse en computadoras y permitir el acceso a usuarios externos, a través de una red local o de Internet, con el fin de recabar información o controlar remotamente a la máquina anfitriona.
- Macro viruses. Virus elaborados mediante un macro lenguaje, por ejemplo, archivos .doc con código malicioso. Finalmente, una vez descargados todos los códigos maliciosos y capturados en archivos pcap se dividieron en dos conjuntos de datos: uno para el entrenamiento (determinar el umbral de *n-grams* sospechosos) y otro para realizar las pruebas de nuestro sistema y comprobar la eficiencia del mismo.

4.2.5. Conclusión

Como se puede observar, la fase de entrenamiento consta de numerosos pasos que el operador humano debe llevar a cabo para completar el entrenamiento del sistema. Todos estos pasos requieren una dedicación de tiempo considerable, además de ser una tarea tediosa y propensa a errores tener que lanzar repetidamente las instrucciones que llevan a cabo cada etapa, teniendo en cuenta que algunas de ellas tienen que repetirse muchas veces. Para ahorrar tiempo y molestias al operario, se han implementado dos programas que automatizan todas las etapas del entrenamiento. Uno de ellos toma las decisiones sobre que reglas, de las generadas en el entrenamiento con cada tipo de datos (tráfico limpio o ataques), es la mejor. El otro automatiza todo el entrenamiento. Solo hay que especificar con la opción adecuada la etapa del entrenamiento que debe ejecutar, así como los parámetros necesarios para usar dicha opción.

4.3. Formato alertas generadas

Además de la integración de las alertas en SDARP, también se dispone de un fichero de registro propio generado tras cada análisis. Dicho fichero está formado por todas las alertas generadas en cada paquete mostrando información de la siguiente manera:

```
-----------------------------------------------------------------
ataque4.pcap
-----------------------------------------------------------------

Paquete analizado: anomalia APAP_preprocessor: payload match =>
fecha:2012-06-05T19:36:11Z
fuente:69.5.88.225 destino:69.5.88.225
puertoO:20480 puertoD:47109 tipo:0 id:43259 proto:6
offset:64 checksum:52197 longitud:54277 datos de apap
[media:84.914062 valork:86.600000 regla:7 ngram:0 diferencia:0.980532]
Paquete analizado: anomalia APAP_preprocessor: payload match =>
fecha:2012-06-05T19:36:11Z
fuente:69.5.88.225 destino:69.5.88.225
puertoO:20480 puertoD:47109 tipo:0 id:43515 proto:6
offset:64 checksum:51941 longitud:54277 datos de apap
[media:86.007812 valork:89.980000 regla:7 ngram:6 diferencia:0.955855]

...etc...
```

Estos son las primeras alertas obtenidas al ejecutar SDARP en modo detección y cargar la traza de un ataque conocido, en nuestro caso el ataque4 de nuestra lista de *datasets* maliciosos. Los parámetros que muestra son

1. Datos del paquete

   fecha: momento en que se produce la alerta en el formato de fechas soportado por el estandar IDMEF es decir, YY:MM:DDThh:mm:ssZ separando la fecha y la hora por 'T' y marcando con una 'Z' el final.

   Fuente: dirección IP del origen

   Destino: dirección IP del destino

   puertoO: puerto del origen

   puertoD: puerto del destino

   tipo: campo tipo del encabezado IP relacionado con la calidad de servicio, es representado en base decimal:

   Bit 0: sin uso, debe permanecer en 0.

   Bit 1: 1 costo mínimo, 0 costo normal.

   Bit 2: 1 máxima fiabilidad, 0 fiabilidad normal.

   Bit 3: 1 máximo rendimiento, 0 rendimiento normal.

   Bit 4: 1 mínimo retardo, 0 retardo normal.

   000xxxxx: de rutina .

   001xxxxx: prioritario.

   010xxxxx: de inmediato.

   011xxxxx: relámpago.

   100xxxxx: invalidación relámpago.

   101xxxxx: procesando llamada crítica y de emergencia.

110xxxxx: control de trabajo de internet.

111xxxxx: control de red.

Id: identificador del paquete

proto: protocolo del paquete

offset: campo offset del encabezado IP

checksum: campo checksum del encabezado IP

longitud: longitud del paquete

2. Datos de SDARP

media: media de apariciones del paquete

valork: valor de la k de la regla que activó la alerta

regla: regla del conjunto de Ks que activó la alerta

n-gram: *n-gram* del paquete en que se activó la alerta

diferencia: porcentaje de similitud entre la media alcanzada cuando saltó la alerta en el paquete y el valor de la k con qué se comparó. Se reserva para usos posteriores

4.3.1. Correlación de alertas

Se ha intentado establecer criterios de correlación de alertas relacionados con la desviación entre la media y el valor de la k considerando los valores extremos, los valores locales y el grado de diferencia pero ninguno ha dado un buen resultado.

Dado que se indica que regla hace saltar la alerta, la mejor opción es insertar las reglas en el conjunto de reglas definitivas considerando el grado de peligrosidad de las trazas de tráfico que las generó, de manera que al saltar la regla, podemos saber el grado de peligro de la anomalía.

Otra opción ha sido la utilización de software externo para llevar a cabo la correlación en función de los datos proporcionados por el encabezado IP del paquete que hizo saltar la alerta. En concreto hemos trabajado con el programa libre ACARM-NG[51] el cual trabaja bajo el formato de entrada estándar IDMEF(rfc4765)[52] de datos de alertas de IDS establecido en el 2007.

4.3.2. Formato de paso de mensajes para detección de intrusiones IDMEF

Intrusion Detection Message Exchange Format (IDMEF). EL objetivo de este formato es poder intercambiar información interesante de las salidas obtenidas por sistemas IDS entre sí, y ser tratada por software externo. Su uso está bastante extendido entre el software de correlación de alertas. Este formato es actualmente utilizado por Snort, Bro y Prelude. En el caso de snort es necesario instalar el IDMEF plug-in para poder generarlo en tiempo real. Por supuesto, las salidas de los preprocesadores que vienen por defecto también son convertidas a este formato.

Para poder trabajar en este formato hemos implementado un módulo suplementario que se encarga de transformar las salidas del registro propio a éĺ. Si desea que se genere

en tiempo real con el resto de salidas de Snort, basta con modificar el código fuente del mencionado plug-in de Snort añadiéndole nuestras funciones de conversión.

El formato IDMEF es una estructura xml[53]. Un ejemplo presentado en su documentación de resultados de traza del ataque ping de la muerte es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<idmef:IDMEF-Message xmlns:idmef="http://iana.org/idmef" version="1.0">
  <idmef:Alert messageid="abc123456789">
    <idmef:Analyzer analyzerid="bc-sensor01">
      <idmef:Node category="dns">
        <idmef:name>sensor.example.com</idmef:name>
      </idmef:Node>
    </idmef:Analyzer>
    <idmef:CreateTime ntpstamp="0xbc71f4f5.0xef449129">
2000-03-09T10:01:25.93464Z</idmef:CreateTime>
    <idmef:Source ident="a1a2" spoofed="yes">
      <idmef:Node ident="a1a2-1">
        <idmef:Address ident="a1a2-2" category="ipv4-addr">
          <idmef:address>192.0.2.200</idmef:address>
        </idmef:Address>
      </idmef:Node>
    </idmef:Source>
    <idmef:Target ident="b3b4">
      <idmef:Node>
        <idmef:Address ident="b3b4-1" category="ipv4-addr">
          <idmef:address>192.0.2.50</idmef:address>
        </idmef:Address>
      </idmef:Node>
    </idmef:Target>
    <idmef:Target ident="c5c6">
      <idmef:Node ident="c5c6-1" category="nisplus">
        <idmef:name>lollipop</idmef:name>
      </idmef:Node>
    </idmef:Target>
    <idmef:Target ident="d7d8">
      <idmef:Node ident="d7d8-1">
        <idmef:location>Cabinet B10</idmef:location>
        <idmef:name>Cisco.router.b10</idmef:name>
      </idmef:Node>
    </idmef:Target>
    <idmef:Classification text="Ping-of-death detected">
      <idmef:Reference origin="cve">
        <idmef:name>CVE-1999-128</idmef:name>
        <idmef:url>http://www.cve.mitre.org/cgi-bin/cvename.cgi?
name=CVE-1999-128</idmef:url>
      </idmef:Reference>
    </idmef:Classification>
  </idmef:Alert>
```

```
</idmef:IDMEF-Message>
```

A continuación mostramos la salida que genero SDARP para las alertas de la traza del ataque4 anteriormente utilizadas de ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<idmef:IDMEF-Message version="1.0" xmlns:idmef="http://iana.org/idmef">
<idmef:Alert messageid= "43259">
<idmef:Analyzer analyzerid= "APAP">
<idmef:Node category= "tcp">
<idmef:location>APAP host</idmef:location>
<idmef:name>www.alkahest.es</idmef:name>
</idmef:Node>
</idmef:Analyzer>
<idmef:Source ident="69.5.88.225">
<idmef:Node ident="69.5.88.225">
<idmef:Address ident="69.5.88.225" category="ipv4-addr">
<idmef:address>69.5.88.225</idmef:address>
</idmef:Address>
</idmef:Node>
<Service>
<port>20480</port>
<protocol>tcp</protocol>
</Service>
</idmef:Source>
<idmef:Target ident="69.5.88.225">
<idmef:Node ident="69.5.88.225" category="dns">
<idmef:name>69.5.88.225</idmef:name>
<idmef:Address ident="69.5.88.225" category="ipv4-addr">
<idmef:address>69.5.88.225</idmef:address>
</idmef:Address>
</idmef:Node>
<Service>
<port>47109</port>
<protocol>tcp</protocol>
</Service>
</idmef:Target>
<idmef:Classification text="deteccion de anomalias">
<idmef:Reference origin="apap">
<idmef:name>deteccion de anomalias</idmef:name>
<idmef:url>www.hostpruebas.es</idmef:url>
</idmef:Reference>
</idmef:Classification>
<AdditionalData type="string" meaning="resultado de APAP">
Paquete analizado: anomalia APAP_preprocessor: payload match
=> fecha:2012-06-05T19:36:11Z
fuente:69.5.88.225
destino:69.5.88.225 puertoO:20480 puertoD:47109
tipo:0 id:43259 proto:6 offset:64
checksum:52197 longitud:54277 datos de apap
```

```
[media:84.914062 valork:86.600000 regla:7 ngram:0 diferencia:0.980532]
</AdditionalData>
<idmef:AdditionalData type ="date-time" meaning="start-time">
<idmef:date-time>2012-06-05T19:36:11Z</idmef:date-time>
</idmef:AdditionalData>
<idmef:AdditionalData type ="date-time" meaning="stop-time">
<idmef:date-time>2012-06-05T19:36:11Z</idmef:date-time>
</idmef:AdditionalData>
</idmef:Alert>

...
..
..   {resto de alertas}
..

</idmef:IDMEF-Message>
```

4.4. Resultados detección

4.5. Optimización del algoritmo

La demanda de mayor velocidad en los distintos tipos de redes, ha obligado a los NIDS a procesar mayor cantidad de tráfico en menos tiempo. En consecuencia, la mayor parte de los fabricantes ha optado por la implementación por hardware, medida que suele conllevar el encarecimiento del coste del producto. El objetivo de nuestros estudios está siendo optimizar el rendimiento de nuestro NIDS[37], basándonos en diferentes técnicas de concurrencia. Esta mejora se basa en aumentar la cantidad de tráfico por unidad de tiempo procesada por el sistema sin necesidad de recurrir a una implementación por hardware. Ha de aclararse, que si bien, estas medidas pueden suponer que nuestro NIDS trabaje en redes más rápidas en tiempo real, actualmente resulta verdaderamente dificil superar el rendimiento de un NIDS implementado por hardware en este aspecto.

A pesar de las virtudes del algoritmo Anagram[38] en cuanto al uso de memoria y tiempo constante para los accesos a la estructura *bloom filter* las primeras pruebas de rendimiento realizadas sobre versiones tempranas de SDARP no fueron satisfactorias, por ello se decidió llevar a cabo una optimización basada en explotar el paralelismo en las actuales arquitecturas multiprocesador.

OpenMP[39] es una API que nos permite añadir concurrencia a las aplicaciones mediante paralelismo con memoria compartida. Se basa en la creación de hilos de ejecución paralelos compartiendo las variables del proceso padre que los crea.

Si comparamos OpenMP con MPI tenemos en común que ámbos estándares se pueden usar en unión con Fortran 77, el Fortran 90, C o C++ para aplicaciones de computación en paralelo. Vale la pena notar que para un grupo de un solo procesador y computadores de multiprocesadores de memoria compartida, es posible usar ambos estándares en el mismo programa de aplicación para realizar un aumento en la potencia del procesamiento. MPI se usa para conectar todas máquinas dentro del grupo para formar una máquina virtual, mientras OpenMP se usa para explotar el paralelismo de la memoria compartida en máquinas individuales de memoria compartida dentro del grupo.

MPI puro	OpenMP puro
Ventajas	**Ventajas**
- Portable a máquinas con memoria distribuida o compartida. - "Escala" más halla de un nodo. - Sin problema de alojamiento de los datos	- Fácil de implementar paralelismo - Baja latencia, alto ancho de banda Comunicación implícita - Balanceamiento dinámico de carga
Desventajas	**Desventajas**
- Difícil de desarrollar y corregir - Alta latencia, bajo ancho de banda - Comunicación explícita - Difícil balancear la carga	- Sólo en máquinas con memoria compartida - "Escala" sólo en un nodo - Posible problema de alojamiento de los datos - Sin orden explícito en los hilos

Comparación OpenMp y OpenMPI

OpenMP comprende tres componentes complementarios:

1. Un conjunto de directivas de compilador usado por el programador para comunicarse con el compilador en paralelismo.

   ```
   #pragma omp <directiva> {<cláusula>}* <\n>
   ```

2. Una librería de funciones en tiempo de ejecución que habilita la colocación e interroga sobre los parámetros paralelos que se van a usar, tal como número de los hilos que van a participar y el número de cada hilo.

3. Un número limitado de las variables de entorno que pueden ser usadas para definir en tiempo de ejecución parámetros del sistema en paralelo tales como el número de hilos.

4.5.1. Resultados de la aplicación de OpenMp

Con la idea de mejorar el ancho de banda en que nuestro NIDS soporta un análisis en tiempo real del tráfico de una red hemos optado por explotar el paralelismo a nivel CPU mediante el uso de las librerías OpenMP.

Como no teníamos muy claro en qué medida paralelizar el algoritmo sin que el coste de cambio de contexto de los hilos perjudique a la optimización, se ha creado cuatro grupos distintos de pruebas correspondientes a cuatro criterios de paralelización distintos.

- Nivel 1: Paralelización total
- Nivel 2: Eliminamos la paralelización de bucles for sencillos.
- Nivel 3: Eliminamos la paralelización de bucles for sencillos de longitud fija Eliminamos la parelelizacion de bucles for sencillos de longitud variable

- Nivel 4 : Eliminamos la paralelización de bucles for sencillos de longitud fija Eliminamos la paralelización de bucles for sencillos de longitud variable Eliminamos iteraciones sobre la variable PRIMEROS

La idea de la construcción de estos grupos está en ir relajando el grado de paralelización empezando por las iteraciones aparentemente más sencillas hasta iteraciones mayores. El equipo utilizado en las pruebas es un intel Core 2 Duo con 4 Gb de memoria Ram, por lo que se ha optado por medir el comportamiento de los grupos trabajando de uno a cuatro hilos. Los resultados completos de las pruebas se encuentran en una anexo al final. Aquí presentamos un breve resumen de los resultados obtenidos en las mismas condiciones y con la misma cantidad de tráfico:

- Tiempo de ejecución sin paralelización:

```
Tiempo Total de Ejecución: 258.2378387451172 milliseconds
numero de hilos: 1
```

- Nivel de paralelización 1:

```
Tiempo Total de Ejecución: 194.9136257171631 milliseconds
numero de hilos: 2

Tiempo Total de Ejecución: 258.5747241973877 milliseconds
numero de hilos: 3

Tiempo Total de Ejecución: 345.6873893737793 milliseconds
numero de hilos: 4
```

- Nivel de paralelización 2:

```
Tiempo Total de Ejecución: 164.6449565887451 milliseconds
numero de hilos: 2

Tiempo Total de Ejecución: 207.6129913330078 milliseconds
numero de hilos: 3

Tiempo Total de Ejecución: 248.7366199493408 milliseconds
numero de hilos: 4
```

- Nivel de paralelización 3:

```
Tiempo Total de Ejecución: 167.8123474121094 milliseconds
numero de hilos: 2

Tiempo Total de Ejecución: 189.3835067749023 milliseconds
numero de hilos: 3

Tiempo Total de Ejecución: 220.137357711792 milliseconds
numero de hilos: 4
```

- Nivel de paralelización 4:

```
Tiempo Total de Ejecución: 164.6437644958496 milliseconds
numero de hilos: 2

Tiempo Total de Ejecución: 221.3263511657715 milliseconds
numero de hilos: 3

Tiempo Total de Ejecución: 228.5993099212646 milliseconds
numero de hilos: 4
```

Vemos como la ejecución en 2 hilos es la que da los mejores resultados. La ganancia frente a la ejecución con paralelizacion total y apoyo de un sólo hilo es la siguiente:

Ganancia de velocidad
speedupNivel1= 258/194 (2 hilos) = 1.32
speedupNivel2= 258/164 (2 hilos) = 1.57
speedupNivel3= 258/167 (2 hilos) = 1.54
speedupNivel4= 258/164 (2 hilos) = 1.57

Resultados de la optimización para los diferentes niveles de paralelización experimentados

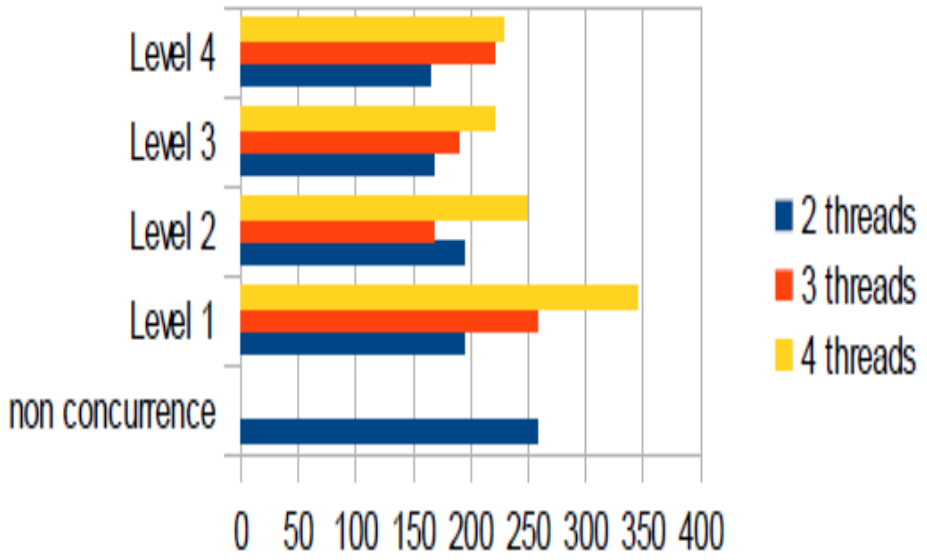

Figura 4.6: Tabla de resultados de Grupos de evaluación de OpemMP

Luego para nuestro Core 2 Duo la mejor opción es explotar el paralelismo a nivel de CPU con dos hilos y aplicando el nivel de paralelización dos. La diferencia entre no emplear el paralelismo a nivel de hilos es considerable: se ha conseguido una mejora del 57 del rendimiento. El siguiente paso sería explotar el paralelismo a nivel de GPU como comentaremos en la sección 5, mediante el uso de librerias openMP[40].

4.6. Comparación con NIDS comerciales

Para está comparación se ha utilizado exclusivamente los valores obtenidos en las métricas de rendimiento en la velocidad de proceso de los datos procedentes de la red. Esto se ha hecho así ya que no existen bancos de pruebas recientes para la evaluación comparativa de IDS con respecto a los valores obtenidos en detección, normalmente se usan los *datasets* sintéticos generados por la DARPA (DARPA98, DARPA99)[41] o el procedente de la KDD cup del 99 (KDD99)[42], sin embargo recurrir a estos conjuntos como elemento de comparación resulta irrelevante puesto que el periodo transcurrido (más de una década) los ha convertido en irrelevantes para evaluar el rendimiento de un IDS actual.

Actualmente el mercado está mayoritariamente marcado por los NIDS por hardware. Esto es debido a que el rápido incremento de las velocidades implica una mayor velocidad de tratamiento de información por parte de los NIDS y en grandes escalas, no resulta viable la implementación por software.

A continuación se muestra una lista de productos NIDS en venta por los fabricantes más punteros y trataremos de comprender hasta que punto puede ser interesante el empleo de nuestro NIDS.

En cuanto a software poco se puede decir. No salen productos nuevos a la venta ya que los NIDS libres como Snort, Prelude[43] o Bro[44] son bastante mejores que los intentos de NIDS comerciales. Uno de los pocos supervivientes ha sido Sax2 NIDS[45] que en su página web promete alcanzar velocidades de hasta 100Mbps pero que en la práctica no llega a alcanzar ni los 50Mbps según su propia comunidad de usuarios. Su precio oscila de 100 dolares de licencia anual a 700 dolares con su licencia global .

Exactamente lo mismo le sucede al NIDS de AthTek NetWalk[46] que con un precio de 500 dolares y prometiendo las mismas velocidades, da exactamente los mismos resultados. Vamos a analizar el precio por velocidad de los NIDS por hardware presentados, de los fabricantes CISCO[47], Palo Alto[48], Juniper[49] y HP[50]:

Podriamos ver que nuestro NIDS trabajando en un Core 2 Duo a una velocidad máxima que como anteriormente vimos corresponde a 12Mbps, tendría un precio aproximado de 125 dolares en el mercado.

El trabajar a dicha velocidad , seria una solución útil para uso domestico o para una pequeña empresa , y probablemente funcionando bajo arquitecturas más modernas como las Intel i5 o i7 y una adecuada selección de las directivas empleadas obtendría un rendimiento mucho mayor.

Modelo	Velocidad (Mbps)	Precio (Dólares)
Cisco 4210	40	500
Cisco 4270	4096	66000
PA-200	50	2000
PA-500	100	5200
PA-5xxx	2048	40000
Juniper IDP75	75	5000
Juniper IDP250	250	19000
Juniper IDP800	800	49000
Juniper IDP8200	8396,8	70000
HP J9521A	300	6000

Figura 4.7: Tabla precio frente a valocidad de procesamineto NIDS

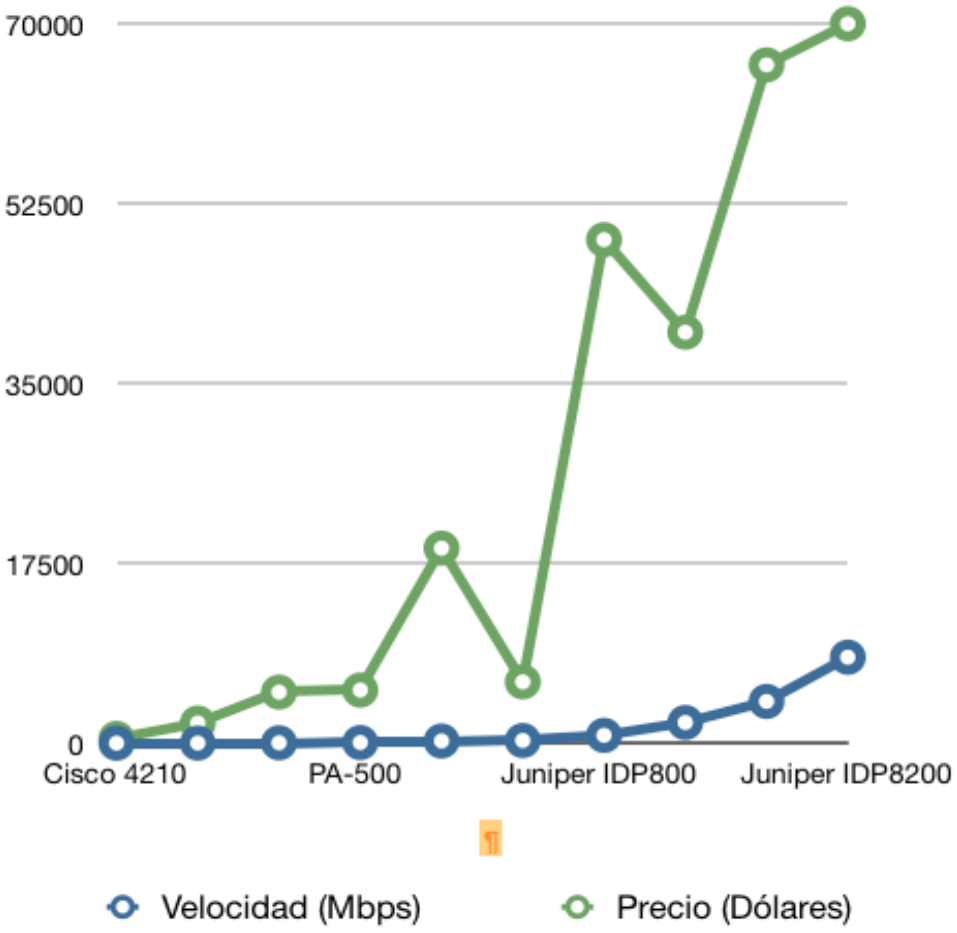

Figura 4.8: Gráfica precio frente a valocidad de procesamineto NIDS

Capítulo 5

Conclusiones y propuestas de trabajo futuro

5.1. Conclusiones

Con el proyecto SDARP finalizado y tras exponer su contexto, bases y el trabajo realizado llega el mometo de evaluar de manera crítica el trabajo realizado.

A grosso modo, opinamos que las metas fijadas al comienzo se han cumplido en su mayor parte de forma satisfactoria. La idea de integrar un preprocesador de anomalías que utilizará aprendizaje automático para la extracción de los modelos de uso normal en el conocido IDS Snort ha resultado un éxito.

Cabe destacar del mismo modo el gran trabajo que ha sido llevado a cabo en la automatización del entrenamiento y el esfuerzo en conseguir mejorar el rendimiento base del algoritmo para una mayor velocidad de proceso.

Así mismo el trabajo de documentación realizado en una temática totalmente desconocida para el equipo de desarrollo como eran al principio del año académico los IDS. Este trabajo ha sido de gran valor cuando no indispensable para la consecución de los objetivos fijados. Cabe destacar en este aspecto la ayuda recibida por parte de los miembros del departamento GASS de la UCM que han aportado valiosa información sobre los métodos y las fuentes válidas de investigación

En definitiva, creemos que el trabajo realizado ha merecido la pena y que el prototipo desarrollado puede ser útil para la protección de una red doméstica o de una pequeña empresa. Así mismo, de forma crítica admitimos que el prototipo desarrollado aún tiene un largo camino que recorrer para convertirse en una herramienta usable tanto a nivel particular como empresarial.

5.2. Propuestas para futuros trabajos

En este apartado se recogen los trabajos que se han considerado de interés por parte de los autores del documento para ser ser llevadas a cabo en el futuro. Se presentan 3 propuestas: la optimización del algoritmo en términos de espacio requerido para el almacenamiento del clasificador, la implementación del citado algoritmo usando un API de

GPGPU (CUDA u OpenCL) para aumentar el rendimiento y por último la integración o implementación de software de correlación de alertas con el fin de disminuir el número de falsos positivos y facilitar el análisis del registro de alertas.

5.2.1. Compresión del clasificador

Uno de los mayores problemas a los que se enfrenta nuestra modificación de Anagram es el gran tamaño de los ficheros que almacenan los datos de los *n-grams*. Para paliar este coste, se diseñó un sencillo algoritmo de compresión de datos que permite reconstruir la estructura original desde el archivo sin causar desgaste en el rendimiento del computador.

La idea radica en que el número de apariciones de la mayoría de elementos puede llegar a ser bastante alto en redes heterogéneas. Para reducir la cantidad de espacio utilizado en disco para almacenar estos datos, podríamos generar un fichero donde los datos no tuviesen una estructura regular de por ejemplo 32 bits por entero, si no que fuésemos capaces de usar únicamente el número de bits necesarios para cada número, y guardando dicha cantidad en un fichero extra del que leeríamos paralelamente a la hora de cargar la estructura.

Primero optimizaríamos las cantidades calculamos la distancia entre posiciones adyacentes del array de apariciones. Esto se consigue manteniendo la última posición del array intacta, y restando al valor de la posición anterior el valor de la siguiente, hasta rellenar una estructura auxiliar. Además rellenaríamos un array auxiliar del mismo número de posiciones, que nos indicaría el número de bits necesarios para leer del fichero resultante dichos valores.

Para cargar la estructura solo habría que leer la cantidad de bits necesarios para leer cada número original, leer esa cantidad en el fichero con los datos, e ir calculando las nuevas posiciones como la suma del contenido de la anterior más el nuevo valor leído.

5.2.2. Optimización del algoritmo mediante GPGPU

GPGPU son las siglas de *General Purpose computing on Graffics Processing Units*, hemos hablado en el apartado 2 de este mismo documento la utilidad de realizar el proceso de detección mediante la GPU. A nuestro parecer es la continuación lógica del trabajo llevado a cabo con la librería OpenMp.

En la realización de este trabajo se plantean varias cuestiones abiertas: alojamiento del *bloom filter* en memoria no reubicable durante el proceso de paginación, para de esa manera garantizar accesos constantes en tiempo sin la interferencia de la paginación. Una posible ubicación para esta estructura es la memoria dedicada de la GPU garantizando de esta forma el acceso más rápido durante la detección. En la fase de entrenamiento por lo tanto no habría problema para ubicarlo en la memoria principal simplificando de esta manera la construcción del clasificador.

El siguiente punto a tratar será la transferencia de los paquetes capturados de la red a la GPU puesto que las operaciones de transferencia a la GPU implican un sobre coste a ser considerado, la solución encontrada en la literatura relacionada ha sido la implementación

de un doble *buffer* de almacenamiento. El primer *buffer* almacenará temporalmente los paquetes recogidos hasta que se llene y envíe los paquetes a la GPU. Los paquetes que lleguen durante el proceso de transferencia serán almacenados en el segundo *buffer*, cuando este esté lleno se repite la operación actuando en este caso el primero como *buffer* de refuerzo.

La asignación de recursos dentro de la GPU es otra de las decisiones que se habrán de tomar puesto que se deberá decidir sobre si dedicar un multiprocesador al completo para procesar cada paquete o por el contrario asignar un procesador de flujo. A juicio de los autores de este trabajo ambas son alternativas viables.

Por último queda tratar como los resultados del análisis serán devueltos a la CPU. El resultado del análisis efectuado sobre cada paquete en concreto será devuelto sobre una posición de un vector que habrá sido previamente alojada en la memoria del dispositivo.

Una ventaja que presenta esta alternativa sobre la tradicional implementación que hace uso de hardware reconfigurable es que los distintos fabricantes de tarjetas gráficas garantizan la retro-compatibilidad de las nuevas versiones de sus productos, sin embargo los fabricantes de hardware reconfigurable no lo hacen, obligando en muchos casos a la re-escritura de los algoritmos para adaptarlos a nuevas versiones del producto.

5.2.3. Utilización o implementación de software de correlación de alertas

Los potenciales beneficios de la utilización de software destinado a la correlación de alertas ya han sido mencionados en este documento. A continuación se enumeran las capacidades de estos sistemas y se apunta brevemente la importancia individual de cada una:

1. Capacidad para reducir el número de alertas: en los actuales IDS multi-detector el número de alertas sobrepasa con creces la capacidad de los operadores para procesar y filtrar las alertas recibidas.

2. Capacidad para agrupar alertas: la capacidad para agrupar las alertas de acuerdo a características comunes sirve para facilitar el análisis de las mismas por parte de los operadores humanos.

3. Capacidad para detectar ataques en múltiples pasos: como consecuencia de la anterior capacidad estos sistemas presentan la capacidad de detectar ataques realizados en múltiples pasos que de otra manera podrían ser descartados durante el análisis y ser tomados como falsos positivos.

4. Capacidad para reducir el número de falsos positivos: al considerar una intrusión como un conjunto de acciones y tener en cuenta la relación entre ellas se pueden descartar gran parte de los eventos aislados que generan falsos positivos

5. Capacidad para detectar ataques conocidos: un sistema de correlación de alertas es útil para detectar ataques conocidos, esto puede ser realizado mediante la especificación de escenarios de ataque o por las precondiciones y consecuencias necesarias para que un ataque sea llevado a cabo.

6. Capacidad para detectar ataques desconocidos: mediante comparación estricta de características comunes entre las alertas generadas es posible detectar ataques de

nueva aparición.

Ya sea mediante la implementación de nuestro propio programa encargado de ello o mediante la adopción de cualquiera de los existentes, mejoraría de manera considerable el desempeño de cualquier sistema de detección de intrusiones mediante la reducción de dos aspectos críticos como son los porcentajes de falsos positivos y falsos negativos así como de ampliar las capacidades de los detectores individuales.

Sin embargo el grupo de desarrolladores propone la implementación del software puesto que ninguna de las técnicas de correlación de alertas investigadas para la realización de este proyecto satisface las seis capacidades arriba enunciadas por completo y resultando de la implementación de un sistema que si lo hiciera un producto novedoso y de aplicación comercial.

Capítulo 6

Documentación de referencia

En este capítulo se incluye la bibiliografía del proyecto SDARP, compuesta por los articulos, libros y páginas web de proveedores o grupos de investigación.

Bibliografía

[1] J.M.Estévez, Pedro García-Teodoro, Jesús E. Díaz-Verdejo, "Anomalydetection methods in wired networks: a survey and taxonomy", Computer Communications 27 pp 1569-1584, 2004.

[2] R. P.Lippmann, R. K.Cunningham, "Improving Intrusion Detection Performance using Keyword Selection and Neural Networks" , Web proceedings of the 2nd International Workshop on Recent Advances in Intrusion Detection, 1999.

[3] Cunningham R. K , Cunningham R. P, "Evaluating Intrusion Detection Systems without Attacking your Friends: The 1998 DARPA Intrusion Detection Evaluation", In Proceedings ID'99, Third Conference and Workshop on Intrusion Detection and Response, San Diego, CA: SANS Institute, 1999.

[4] McHugh.J, "Testing intrusion detection systems: a critique to the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory", ACM Transactions on Information and System Security 3 pp 262–294, 1999

[5] Rash.M, "Intrusion Prevention and Active Response: Deploying Network and host IPS",Syngress , 2005

[6] Security-Enhanced Linux http://www.nsa.gov/research/selinux/index.shtml, 26/Jun/2012

[7] J.P.Anderson Co, "Computer Threat Monitoring and Surveillance", In Anderson, J.P. Technical report, 1980.

[8] Steven. L. Scott, "A Bayesian paradigm for designing intrusion detection system",Computational Statistics & Data Analysis 45 pp 69-83, 2004.

[9] J.M. Estévez-Tapiador , P.García-Teodoro, J.E. Daz- Verdejo,*Measuring normality in HTTP traffic for anomaly-based intrusion detection* ,Computer Networks 45 pp 175-193, 2004.

[10] M.Bishop, "A taxonomy of (Unix) system and network vulnerabilities, Technical Report CSE-9510, Department of Computer Science, University of California at Davis, May 1995.

[11] J.D. Howard, T.A. Longstaff, "A common language for computer security incidents", Technical report, Sandia National Laboratories, 1998.

[12] D.L Lough,"A taxonomy of computer attacks with applications to wireless networks", PhD thesis, Virginia Polytechnic Institute and State University; 2001.

[13] CERT, "CERT Coordination Center Statistics", http://www.cert.org/stats/', 2008.

[14] Anti-phishingworking group, "phishing activity trends report, 4th quarter / 2009 ", http://www.antiphishing.org/reports apwg report Q4 2009.pdf, 2010.

[15] P. Porras, D.Schnackenberg, "S.Staniford-Chen, The Common Intrusion Detection Framework Architecture", http://gost.isi.edu/cidf/,1999.

[16] ISO/IEC:18043:2006.

[17] Rich Feiertag, Cliff Kahn, "PhilPorras, Dan Schnackenberg, Stuart Staniford-Chen, A Common Intrusion Specification Language (CISL)", http://gost.isi.edu/cidf/,1999.

[18] H. Debar, D.Curry, "B.Feinstein,The Intrusion Detection Message Exchange Format, RFC 4765", 2007.

[19] I. Perona, O. Arbelaitz, I. Gurrutxaga, J. Martn, J. Muguerza, Jesús, M. Pérez, "Unsupervised, Anomaly Detection System For Nids-S Based, On Payload And Probabilistic Suffix Trees", IADIS International Conference Applied Computing pp 11-18, 2009.

[20] I. Perona, O. Arbelaitz, I. Gurrutxaga, J. Martn, J. Muguerza, J.M. Pérez, "Service-independent payload analysis to improve intrusion detection in network traffic". Proc. of the 7th Australasian Data Mining Conference, 2008.

[21] S.J.Stolfo, W.Lee, P.K.Chan,"Data Mining-based Intrusion Detectors: An Overview of the Columbia IDS Project",ACM SIGMOD Record Volume 30 Issue 4, pp 5-14, 2001.

[22] U. Lindqvist, P.Porras, "Detecting computer and network misuse through the production-based expert system toolset (P-BEST)",In Proceedings of the 1999 IEEE Symposium on Security and Privacy, pp 146-161,1999.

[23] P. Neumann, P. Porras, "EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances", Proceeding of 20th National Information pp 353-365, 1997.

[24] M. Roesch," SNORT — Ligthweigth Intrusion Detection for networks ", Proceedings of LISA '99: 13th Systems Administration Conference pp 229-238,1999.

[25] U. Zurutuza, "Estado del arte Sistemas de Detección de intrusos",mondragon.edu,2004.

[26] S.L. Scott, P. Smyth ,"The Markov modulated Poisson process and Markov Poisson cascade with applications to web traffic data" ,Bayesian statistics 7 pp 671-680, 2003.

[27] W.Lee, S.J. Stolfo, K.W. Mok, "A Data Mining Framework for Adaptive Intrusion Detection", 1999. Proceedings of the 1999 IEEE Symposium on Security and Privacy pp 120-132, 1999.

[28] D. Barbará, J.Couto, S.Jajodia, N. Wu, "ADAM: A Testbed for Exploring the Use of Data Mining in Intrusion Detection", ACM SIGMOD Record Volume 30 pp 15-24, 2001.

[29] B. Morin, L. Me, H. Debar, and M.Ducasse," M2D2: A Formal Data Model for IDS Alert Correlation, RAID'02 Proceedings of the 5th international conference on Recent advances in intrusion detection pp 115-137, 2002".

[30] David Wagner,Paolo Soto,"Mimicry Attacks on HostBased,Intrusion Detection Systems",Proceedings of the 9th ACM conference on Computer and communications security,2002

[31] Lindqvist U, "Jonsson E. How to systematically classify computer security intrusions", IEEE Security and Privacy, 1997.

[32] F. Cuppens. "Managing Alerts in a Multi-Intrusion Detection Environment". In 17th Annual Computer Security Applications Conference (ACSAC) New-Orleans, New-Orleans, USA, December 2001.

[33] [F.Cuppens, A.Miege, "Alert Correlation in a Cooperative Intrusion Detection Framework", Proceedings of IEEE Symposium Security and Privacy, 2002

[34] A.Valdes, K.Skinner, "Probablistic Alert Correlation". Proceedings of the Recent Advances in Intrusion Detection (RAID). Davis, CA, 2001

[35] X.Qin, W.Le. "Statistical Causality of INFOSEC Alert Data", proceedings of Recent Advances in Intrusion Detection, 2003.

[36] About GPGPU, http://gpgpu.org/about, 26/Jun/2012

[37] "Malware Detection System by Payload Analysis of Network Trafficc"submitted to RAID 2012

[38] K. Wang, J. J. Parekh, S. Stolfo: Anagram: A content Anomaly Detector Resistant to Mimicry Attack. Recent Advances in Intrusion Detection, Springer, Vol. 4219, pp 226-248, (2006)

[39] OpenMP home page [online] available: http://openmp.org

[40] OpenCL home page [online] available: http://www.khronos.org/opencl

[41] Brugger ST, and Chow J (2005) An Assessment of the DARPA IDS Evaluation Dataset Using Snort. Available via UCDAVIS department of Computer Science. http://www.cs.ucdavis.edu/research/tech-reports/2007/CSE-2007-1.pdf. Cited 2 May 2007

[42] KDD cup 1999 [online] available: http://www.kdd.org/kdd-cup-1999-computer-network-intrusion-detection

[43] Prelude IDS home page [online] available: http://www.prelude-technologies.com

[44] BRO IDS home page [online] available: http://bro-ids.org/

[45] SAX2 IDS home page [online] available: http://sourceforge.net/projectssax2.0

[46] Athtek netwalk home page [online] available: http://www.athtek.com/netwalk.html

[47] Cisco home page [online] http://www.cisco.com/web/ES/index.html

[48] Palo Alto home page [online] http://www.paloaltonetworks.com/

[49] Juniper home page [online] http://www.juniper.net

[50] HP home page [online] http://www.hp.com

[51] ACARM-ng home page [online] http://www.acarm.wcss.wroc.pl/

[52] IDMEF rfc 4765 [online] http://www.ietf.org/rfc/rfc4765.txt

[53] IDMEF ejemplos [online] http://en.wikipedia.org/wiki/IDMEF

Apéndices

Apéndices A

Preprocesadores de Snort

A.1. FRAG3

FRAG3 sustituye al antiguo FRAG2. Su objetivo principal es reensamblar paquetes fragmentados para tratar de evitar ataques de evasión basados en fragmentación como las técnicas basadas en ambigüedades en las politicas de superposición o las basadas en la explotación del campo TTL de la cabecera de los paquetes. De esta forma, podremos pasar el paquete entero por el modulo de detección al igual que lo haría el sistema protegido. Las directivas de configuración de FRAG3 son: `frag3_global` y `frag3_engine`.

A.1.1. Frag3_global

`Frag3_global` permite establecer el número máximo de paquetes simuntáneos a analizar (`max_frags`) y la memoria máxima que puede utilizar (memcap). Por otro lado, también nos permite fijar el número de fragmentos individuales a analizar (`prealloc_frags`).

A.1.2. Frag3_engine

`Frag3_engine` a diferencia de `frag3_global`, permite establecer distintas directivas del motor. Considerese a `frag3_global` como una directiva por general. Podremos configurar el tiempo que permanece el fragmento antes de ser eliminado (timeout), el intervalo de valores TTL en que trabajará (`ttl_limit` y `min_ttl`).También puede tratar de detectar determinadas anomalias en los fragmentos (`detect_anomalies`)o establecer una lista negra de direcciones IP a analizar (`bind_to`) que, por supuesto, serán todas las direcciones por defecto. Para evitar ataques de superposición podemos indicarle la política a seguir (policy), la cual se encontrará en la siguiente lista: bsd, last, firs, linux y BSD-right .Tendremos que elegir la que corresponda al sistema a proteger.

- Ejemplo:

```
preprocessor frag3_engine: policy first, bind_to [192.168.2.0/24,]
```

A.2. Stream5

Al igual que su predecesor, Stream4, este preprocesador permitirá detectar ataques basados en el estado de la conexión TCP, que tratarán de hacer pasar a través de el NIDS

el ataque con fragmentos de relleno que el sistema a proteger descartará por ser incorrectos por causas como pertenecer a una secuencia fuera de ventana o RST/FIN incorrectos. Stream5 podrá guardar paquetes que permiten reconstruir e identificar una sesión completa en caso necesario. Stream5 se puede configurar de cuatro maneras distintas: global, TCP, ICMP, UDP.

En la configuración global entre otras cosas, podemos activar o desactivar el seguimiento de sesiones (`track_tcp`), así como asignar el numero máximo permitido y la memoria reservada (`mem_tcp`y `memcap`).Analogamente podemos hacer lo mismo para los demás protocolos anteriormente indicados, con comandos del tipo (`track_icmp`, `mem_udp`etc).

- Ejemplo de configuración:

```
preprocessor stream5_global: /
max_tcp 9453, track_tcp yes, track_udp yes, track_icmp no
```

En los otros modos tendremos parametros específicos. A modo de ejemplo, la configuración tcp permite activar la detección de robo de sesiones (`check_session_hijacking`) o la detección de anomalías sobre dicho protocolo (`detect_anomalies`).En ICMP y UDP, por ejemplo podemos establecer el timeout del fragmento (`timeout`).

A.3. sfPortScan

El objetivo de *portscan* no es detectar el ataque en si, sino detectar algunos de los métodos mas utilizados para enumerar el sistema victima. sfPortScan ha sido diseñado para detectar distintas maneras de obtener información de un sistema, como *scan* UDP, *scan* TCP, *scan* IP, Decoys, Sweep Scan y muchas más. Este preprocesador se ha especializado en detectar los procedimientos de una de las herramientas de enumeración mas utilizadas: NMAP.

Es necesario tener activado Stream5 para su correcto seguimiento de determinadas sesiones, como las establecidas por el protocolo UDP.

- Al configurar sfPortScan elegiremos los protocolos a inspeccionar:

```
proto tcp | udp | ip | all
```

- Memoria que le asignaremos:

```
memcap
```

- Sensibilidad:

```
sense_level  low|medium|high
```

- Tipo de *scan* a detectar:

```
scan  type  [all | decoy_portscan | distributed_portscan|]
detect_ack_scans
include
midstream
...
```

- Nombre de fichero de logs (`logfile`)o datos como listas blancas (`ignore_scanners`)entre otros parametros.
- Ejemplo de configuración:

```
preprocessor sfportscan: proto { all } scan_type { all } memcap /
{ 10000000 } sense_level { medium } logfile { portscan.log } /
ignore_scanners { 192.168.1.107, 192.168.1.108 }
/ignore_scanned { 192.168.100.0/24}
```

A.4. RPC DECODE

Llamamos flujo RPC (del inglés *Remote Procedure Calls*) al flujo de llamadas a procedimientos remotos. Este preprocesador normaliza múltiples registros fragmentados RPC en un único paquete, para que sean más fáciles de analizar. Si el preprocesador stream5 está inicializado, solamente analizará el tráfico del lado del cliente. Por defecto trabaja sobre los puertos 111 y 32771. El tratamiento de las alertas podrá realizarse a nivel individual, o bien a nivel del paquete ensamblado que junta varias llamadas.

A.5. Performance Monitor

Performance Monitor tiene una finalidad muy distinta al resto de preprocesadores visto hasta el momento: medir el funcionamiento en tiempo real y teórico máximo de snort, con el fín de analizar hasta qué punto nuestro NIDS está funcionando en tiempo real. Es necesario tener algún modo de salida habilitado, ya sea mediante un fichero o la propia terminal, para poder observar los resultados.

Este preprocesador viene activado por defecto. Algunos de los parámetros que mide son la velocidad a la que trabaja (Mbps), el numero de descartes de paquetes total y por unidad de tiempo, alertas por segundo, intentos de conexión y conexiones establecidas por segundo, el estado de los *buffers* de entrada y salida... y así podríamos completar una lista de más de 100 datos que muestrea.

También dispone de varias opciones de ejecución como `events` que muestra las reglas que se han evaluado, `max` que presupone snort trabajando a su máximo teórico para recalcular datos, `console` activa la salida de datos por consola, `flow-ip` muestra estadísticas basadas en trafico entre hosts, dos a dos, `pktcnt` que muestra los paquetes tratados antes de mostrar resultado

- Ejemplo de configuración:

```
preprocessor perfmonitor: \
time 30 events flow file stats.profile max console pktcnt 10000
```

A.6. HTTP Inspect

HTTP (Protocolo de Transferencia de hipertexto) es el protocolo usado en cada transacción de la WWW. HTTP es un protocolo sin estado, es decir, que no guarda ninguna información sobre conexiones anteriores. El desarrollo de aplicaciones web necesita frecuentemente mantener estado. Para esto se usan las *cookies*, que es información que un servidor puede almacenar en el sistema cliente. Esto le permite a las aplicaciones web instituir la noción de "sesión", y también permite rastrear usuarios ya que las *cookies* pueden guardarse en el cliente por tiempo indeterminado.

HTTP Inspect un modulo que proporciona la capacidad de examinar el tráfico HTTP, incluyendo URL's, *cookies*, parámetros por GET/POST, codificación de caracteres, etc. Tiene la capacidad de detectar ataques realizados con herramientas como Nikto y Nessus.

Para habilitar este preprocesador se tienen dos módulos,

- `http_inspect global` Módulo para configuración global.
- `http_inspect server` Modulo de configuración especifica de cada servidor web en el segmento de red.

Http_inspect global como su nombre indica, determina la funcionalidad global de HTTP Inspect, las opciones disponibles que destacan por su utilidad son:

- `iis_unicode_map` para indicar como ha de decodificar caracteres unicode en acuerdo con es sistema a proteger, *proxy alert* que avisa sobre el empleo de proxies preconfigurados, o
- `compress/decompress_depth` para indicar la cantidad maxima de datos a obtener de un paquete comprimido.

Por otro lado, el modo servidor permite configurar algunos parámetros como por ejemplo con `profile` especificaremos la versión del servidor web que queremos proteger, `ports` para el puerto, `enable_cookies` para habilitar trabajar con *cookies*, ascii para indicar cuando decodificar caracteres ascii, `no alerts` no muestra alertas aunque sí serán tenidas en cuenta por el módulo detector de snort y bastantes mas.

- Ejemplo:

```
preprocessor http_inspect: \
global \
iis_unicode_map <map_filename> \
codemap <integer> \
[detect_anomalous_servers] \
[proxy_alert] \
[max_gzip_mem <num>] \
[compress_depth <num>] [decompress_depth <num>] \
[memcap <num>] \
disabled
```

A.7. Preprocesador SMTP

Simple Mail Transfer Protocol (SMTP) Protocolo Simple de Transferencia de Correo, es un protocolo de la capa de aplicación . Protocolo de red basado en textos utilizados para el intercambio de mensajes de correo electrónico entre computadoras u otros dispositivos (PDA's, teléfonos móviles, etc.).

El SMTP preprocesador funciona como un decodificador. Se le pasa un buffer con la carga útil del paquete y lo analiza buscado comandos SMTP. Lo clasificará en comandos de cabecera y contenido TLS. También mantendrá el estado en que se encuentra la relación cliente-servidor.

En su configuración podemos establecer el puerto con el que queramos trabajar(`ports\{25,60,...\}`) y si queremos que conserve el estado (`stateful`/ `stateless`). Podemos establecer si queremos que el análisis considere los espacios entre comandos (`normalizeall`/ `none`/ `cmds`). Se puede decidir si analizar los datos o solamente los encabezados (`ignore_data`),o no analizar contenido codificado(`ignore_tls_data`).

Podemos pasarle una lista de los comandos que queremos que considere como válidos(`valid_cmds\{\}`/ `invalid_cmds\{\}`). Así como diversas configuraciones acerca de la codificación empleada, el tipo de alertas mostradas, listas blancas de direcciones, etc.

- Ejemplo:

```
preprocessor SMTP: \
ports { 25 } \
inspection_type stateful \
normalize cmds \
normalize_cmds { EXPN VRFY RCPT } \
alt_max_command_line_len 260 { MAIL } \
alt_max_command_line_len 300 { RCPT } \
alt_max_command_line_len 500 { HELP HELO ETRN } \
alt_max_command_line_len 255 { EXPN VRFY }
```

A.8. Pop

Post Office Protocol (POP3, Protocolo de la oficina de correo) en clientes locales de correo para obtener los mensajes de correo electrónico almacenados en un servidor remoto. Es un protocolo de nivel de aplicación.

Las versiones del protocolo POP (informalmente conocido como POP1) y POP2 se han hecho obsoletas debido a las últimas versiones de POP3. En general cuando uno se refiere al término POP, nos referimos a POP3 dentro del contexto de protocolos de correo electrónico.

POP es un preprocesador que trabaja sobre POP3 a nivel de capa de aplicación. Dado un buffer de entrada, POP buscará comandos o reglas POP3, y las extraerá descodificándolas de manera apropiada. Este preprocesador almacena el estado entre entre paquetes individuales, manteniendo así una correcta asociación independientemente de los

cambios en el servidor. Es importante activar Stream5 al usar POP, ya que los paquetes deberán de pasar por un correcto ensamblado.

La configuración permite:

- fijar puertos `ports//`
- codificaciones y modo de trabajo

```
b4_decode_depth
qp_decode_depth
...
```

- espacio asignado `memcap`.
- Ejemplo

```
preprocessor pop: \
ports { 110 } \
memcap 1310700 \
qp_decode_depth -1 \
b64_decode_depth 0 \
bitenc_decode_depth 100
```

A.9. FTP / TELNET

FTP (siglas en inglés de *File Transfer Protocol*, Protocolo de Transferencia de Archivos en español) en informática, es un protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP (Transmission Control Protocol), basado en la arquitectura cliente-servidor. Desde un equipo cliente se puede conectar a un servidor para descargar archivos desde él o para enviarle archivos, independientemente del sistema operativo utilizado en cada equipo.

Telnet (TELecommunication NETwork) es el nombre de un protocolo de red a otra máquina para manejarla remotamente como si estuviéramos sentados delante de ella. También es el nombre del programa informático que implementa el cliente. Para que la conexión funcione, como en todos los servicios de Internet, la máquina a la que se acceda debe tener un programa especial que reciba y gestione las conexiones.

FTP/TELNET preprocesador trabaja extrayendo comandos y reglas de un buffer de entrada, tanto de FTP como de Telnet. Guardará información del estado entre los dos extremos de la comunicación, ya que ambos son protocolos orientados a conexión. Disponemos de 4 posibles modos de configuración: Global, Telnet , cliente FTP y servidor FTP. La configuración global es común a todos e implementa las siguientes opciones

- permitir conservar el estado

```
inspection_type statefull /stateless
```

- tratar el tráfico codificado

`encrypted trafic yes/no`

- Ejemplo:

```
preprocessor ftp_telnet: \
global inspection_type stateful encrypted_traffic no
```

1. La configuración Telnet da parámetros propios del protocolo Telnet. Podemos definir puertos (`port`), si normalizar el texto eliminando caracteres delimitadores (`normalize`)o detectar anomalías (`detect_anomalies`).

   - Ejemplo:

   ```
   preprocessor ftp_telnet_protocol: \
   telnet ports { 23 } normalize ayt_attack_thresh 6
   ```

2. La configuración Telnet da parámetros propios del protocolo Telnet.

   - Definir puertos `port`.
   - normalizar el texto eliminando caracteres delimitadores `normalize`.
   - detectar anomalias `detect_anomalies`.
   - Ejemplo:

   ```
   preprocessor ftp_telnet_protocol: \
   telnet ports { 23 } normalize ayt_attack_thresh 6
   ```

3. La configuracion FTP Server puede llevarse a cabo por defecto o fijando una IP

   - Podemos configurar listas de comandos válidos: `ftp_cmds`.
   - Podemos configurar puertos permitidos: `ports`.
   - Podemos hacer que avise de comandos telnet dentro de ftp: `telnet_cmds`.
   - Ejemplo

   ```
   preprocessor _telnet_protocol: \
   ftp server 10.1.1.1 ports { 21 } ftp_cmds { XPWD XCWD }
   ```

4. FTP Client tambien puede iniciarse como default o fijando una dirección IP. Tiene opciones especificas como la de detectar bounce atacks (`bounceyes/no`) o habilitar la inspección de comandos telnet dentro de FTP (`telnet_cmd yes/no`).

   - Ejemplo:

   ```
   preprocessor ftp_telnet_protocol: \
   ftp client default \
   max_resp_len 256 \
   bounce yes \
   telnet_cmds yes
   ```

A.10. SSH

SSH (*Secure SHell* , en español: intérprete de órdenes segura) es el nombre de un protocolo y del programa que lo implementa, y sirve para acceder a máquinas remotas a través de una red. Permite manejar por completo la computadora mediante un intérprete de comandos, y también puede redirigir el tráfico de X para poder ejecutar programas gráficos si tenemos un Servidor X (en sistemas Unix y Windows) corriendo.

Además de la conexión a otros dispositivos, SSH nos permite copiar datos de forma segura (tanto ficheros sueltos como simular sesiones FTP cifradas), gestionar claves RSA para no escribir claves al conectar a los dispositivos y pasar los datos de cualquier otra aplicación por un canal seguro tunelizado mediante SSH.

Este preprocesador detecta algunos de los ataques más conocidos contra este protocolo : *challenge response buffer overflow*, *CRC32*, *Secure CRT* y el *Protocol Mismatch exploit*.

Podemos configurar éste preprocesador a distintos niveles:

- Nivel de puertos de trabajo: `server_ports\{\}`.
- Nivel de máximo de paquetes por cliente: `max_encrypted_packets`.
- Nivel de máximo de bytes por cliente: `max_client_bytes`.
- Detección de los ataques mencionados:

```
enable_respoverflow
enable_ssh1crc2
enable_srvoverflow
...
```

- Ejemplo de configuración:

```
preprocessor ssh: \
server_ports { 22 } \
max_client_bytes 19600 \
max_encrypted_packets 20 \
enable_respoverflow \
Enable_ssh1crc32
```

A.11. DNS

Domain Name System o DNS (en español: sistema de nombres de dominio) es un sistema de nomenclatura jerárquica para computadoras, servicios o cualquier recurso conectado a Internet o a una red privada. Este sistema asocia información variada con nombres de dominios asignado a cada uno de los participantes. Su función más importante, es traducir (resolver) nombres inteligibles para los humanos en identificadores binarios asociados con los equipos conectados a la red, esto con el propósito de poder localizar y direccionar estos equipos mundialmente.

El servidor DNS utiliza una base de datos distribuida y jerárquica que almacena información asociada a nombres de dominio en redes como Internet. Aunque como base de datos el DNS es capaz de asociar diferentes tipos de información a cada nombre, los usos más comunes son la asignación de nombres de dominio a direcciones IP y la localización de los servidores de correo electrónico de cada dominio.

DNS Preprocesador analiza tráfico del protocolo DNS en los paquetes TCP y UDP en busca de vulnerabilidades conocidas como DNS Client RData Overflow, Obsolete Record Types, and Experimental Record Types. Requiere la activación de Stream5.

Alguna de las opciones de configuración que presenta:

- Selección de puertos: `ports{}` .
- Alertas sobre registros antiguos: `enable_obsolete_types`.
- Experimentales: `enable_experimental_types`.
- Detectar determinadas vulnerabilidades: `enable_rdata_overflow`, etc.
- Ejemplo:

```
preprocessor dns: \
ports { 53 } \
enable_rdata_overflow
```

A.12. SSL / TLS

Secure Sockets Layer (SSL) en español capa de conexión segura y su sucesor *Transport Layer Security* (TLS) en español seguridad de la capa de transporte son protocolos criptográficos que proporcionan comunicaciones seguras por una red, comúnmente Internet.

SSL proporciona autenticación y privacidad de la información entre extremos sobre Internet mediante el uso de criptografía. Habitualmente, sólo el servidor es autenticado (es decir, se garantiza su identidad) mientras que el cliente se mantiene sin autenticar. Normalmente éste tipo de tráfico fluye por el puerto 443 bajo el protocolo HTTPS.

El tráfico encriptado tiende a generar un alto número de falsos positivos en Snort. Éste preprocesador además de decodificar la parte perteneciente a cualquiera de los protocolos anteriormente mencionados, sirve para determinar cúando serán analizados.
Podemos configurar los puertos a analizar (`Ports\{\}`),deshabilitar la inspección de tráfico encriptado (`noinspect_encrypted`) o mejorar el rendimiento deshabilitando el requisito de observar los dos lados de la comunicación para conseguir un mejor rendimiento (`trustservers`).

- Ejemplo de configuración:

```
preprocessor ssl: noinspect_encrypted
```

A.13. Preprocesador ARP Spoof

Como ya se explicó en las técnicas de evasión de NIDS, El ARP Spoofing, también conocido como ARP Poisoning o ARP Poison Routing, es una técnica usada para infiltrarse en una red Ethernet conmutada (basada en switch y no en hubs), que puede permitir al atacante husmear paquetes de datos en la LAN (red de área local), modificar el tráfico, o incluso detenerlo.

El principio de la falsificación ARP es enviar mensajes ARP falsos (falsificados, o spoofed) a la Ethernet. Normalmente la finalidad es asociar la dirección MAC del atacante con la dirección IP de otro nodo (el nodo atacado), como por ejemplo la puerta de enlace predeterminada (*gateway*). Cualquier tráfico dirigido a la dirección IP de ese nodo, será erróneamente enviado al atacante, en lugar de a su destino real. El atacante, puede entonces elegir, entre reenviar el tráfico a la puerta de enlace predeterminada real (ataque pasivo o escucha), o modificar los datos antes de reenviarlos (ataque activo). El atacante puede incluso lanzar un ataque de tipo DoS (Denegación de Servicio) contra una víctima, asociando una dirección MAC inexistente con la dirección IP de la puerta de enlace predeterminada de la víctima.

Éste preprocesador decodificar los paquetes ARP en busca de ataques ARP, Unicast, petición ARP e inconsistencias en el la topología de la red.

Si se lanza sin argumentos, solamente inspeccionará las direcciones ARP de los paquetes de la red en busca de irregularidades. Si especificamos `-unicast`buscará este tipo de solicitudes y avisará con alertas. Con `arpspoof_detect_host`, y especificando un par de direcciones IP y MAC para usarlas en el caso de estar ante una ataque de sobrescritura de caché ARP.

- Ejemplo:

```
preprocessor arpspoof[: -unicast]
preprocessor arpspoof_detect_host: 192.168.40.1 f0:0f:00:f0:0f:00
```

A.14. DCE / RPC 2

Server Message Block o SMB es un Protocolo de red (que pertenece a la capa de aplicación en el modelo OSI) que permite compartir archivos e impresoras (entre otras cosas) entre nodos de una red. Es utilizado principalmente en ordenadores con Microsoft Windows y DOS.

Los servicios de impresión y el SMB para compartir archivos se han transformado en el pilar de las redes de Microsoft. Con la presentación de la Serie Windows 2000 del software, Microsoft cambió la estructura subyacente para el uso del SMB. En versiones anteriores de los productos de Microsoft, los servicios de SMB utilizaron un protocolo que no es TCP/IP para implementar la resolución de nombres de dominio. Comenzando con Windows 2000, todos los productos subsiguientes de Microsoft utilizan denominación DNS. Esto permite a los protocolos TCP/IP admitir directamente el compartir recursos SMB.

DCE *Remote Procedure Call* o bien DCE RPC es un sistema de llamada a procedimiento remoto del conjunto de software OSF DCE. DCE / RPC, la abreviatura de *Distributed Computing Environment / Remote Procedure Calls*, es el sistema de llamada

a procedimiento remoto desarrollado para el entorno de la informática distribuida (DCE). Este sistema permite a los programadores escribir software distribuido como si fuera todos los que trabajan en el mismo equipo, sin tener que preocuparse por el código de red subyacente. El preprocesador DCE/RPC 2 tiene como objetivo facilitar la detección de técnicas de evasión como la desegmentación SMB o fragmentación DCE/RPC.

SMB desegmentation actúa sobre los siguientes comandos que intervienen en el transporte peticiones DCE/RPC y respuestas: *Write, Write Block Raw, Write and Close, Write AndX, Transaction, Transaction Secondary, Read, Read Block Raw* y *Read AndX*. Dicho transporte puede llevarse acabo por los protocolos SMB, UDP, TCP, RPC y sobre HTTP v1 Server y Client. El objetivo además es reducir la tasa de falsos positivos t el coste y complejidad de las reglas anteriores parar DCE/RPC.

Requiere de Stream5 activo con un *session tracker* para guardar datos. El re ensamblado de Stream5 ha de analizar las sesiones TCP. La desfragmentación IP tiene que estar activa. También tendrá que estar activo Frag3

DCE/RPC preprocessor tiene dos configuraciones:

1. Global `dcerpc2` la configuración global permite utilizar parámetros similares a otros preprocesadores en este modo:
   - `mempcap`.
   - `disable_defrag`.
   - `events`.
   - `disabled`.
2. Orientada a servidor `dcerp2_server`en lado del servidor dispone de:
   - configuraciones propias de la red `net`.
   - politicas `policy`.
   - parametros exclusivos como:

     ```
     autodetect
     no_autodetect http_proxy_ports
     smb_max_chain
     ...
     ```

- Ejemplo:

```
preprocessor dcerpc2_server: \
net [10.4.10.0/24,feab:45b3::/126], policy WinVista, \
detect [smb, tcp, rpc-over-http-proxy 8081],
autodetect [tcp, rpc-over-http-proxy [1025:6001,6005:]], \
smb_invalid_shares ["C$", "ADMIN$"], no_autodetect_http_proxy_ports
preprocessor dcerpc2: memcap 102400
```

A.15. Sensitive Data Preprocesor

Información personal, información personalmente identificable o información personal de identificación (del inglés *Personally Identifiable Information* (PII)), es un concepto utilizado en seguridad de la información. Se refiere a la información que puede usarse para identificar, contactar o localizar a una persona en concreto, o puede usarse, junto a otras fuentes de información para hacerlo. Se utiliza muy extensamente la abreviatura PII. Las definiciones legales, especialmente en el contexto del derecho al honor y la intimidad o privacidad, varían en cada país. Son informaciones personales habitualmente demandadas o protegidas el nombre, el domicilio, el número de identificación personal en sus distintas formas, el número de teléfono, la dirección IP (en algunos casos), el documento de identidad, el número del carnet de conducir.

La función de este preprocesador es detectar dicho tipo de información y proceder a su filtrado de la manera especificada. Es importante que trabaje con Stream5 activo.

La configuración del preprocesador consta de dos partes: por un lado se configuran los parametros y por otro lado las reglas. Algunos parámetros sirven, por ejemplo, para avisar de PII en la carga útil(`alert_threshold`), marcar con X los 4 últimos dígitos de tarjetas de crédito (`mask_output`) o el tipo de agrupamiento de números de la seguridad social americana a usar (`ssn_file`).

- Ejemplo

```
preprocessor sensitive_data: alert_threshold 25 \
mask_output \
ssn_file ssn_groups_Jan10.csv
```

Por otro lado, las reglas corresponden a los patrones de representación de cada dato en función de la nacionalidad. Es un campo bastante extenso y no entraremos en detalles por su escaso interes para este apéndice.

A.16. Normalizer

Cuando trabajamos con Snort en modo inline, realizaremos tareas de IPS(*Intrusion Prevention Sistem*), es decir, de prevención de intrusiones. Además de las labores del IDS, el IPS podrá realizar labores activas para bloquear/desbloquear un determinado tráfico sobre una determinada región de nuestra red.

Las técnicas de normalización de paquetes que facilita este preprocesador permite minimizar las opciones de evasión de tráfico ilícito. Dicha normalización será especifica para el tipo de paquete: IP4, IP6, ICMP4/6, TCP y TTL

A.17. Preprocesador SIP

Session Initiation Protocol (SIP o Protocolo de Inicio de Sesiones) es un protocolo desarrollado por el grupo de trabajo MMUSIC del IETF con la intención de ser el estándar para la iniciación, modificación y finalización de sesiones interactivas de usuario donde

intervienen elementos multimedia como el video, voz, mensajería instantánea, juegos en línea y realidad virtual.

La sintaxis de sus operaciones se asemeja a las de HTTP y SMTP, los protocolos utilizados en los servicios de páginas Web y de distribución de e-mails respectivamente. Esta similitud es natural ya que SIP fue diseñado para que la telefonía se vuelva un servicio más en Internet.

En noviembre del año 2000, SIP fue aceptado como el protocolo de señalización de 3GPP y elemento permanente de la arquitectura IMS (IP Multimedia Subsystem). SIP es uno de los protocolos de señalización para voz sobre IP, otro es H.323 y IAX actualmente IAX2.

ISP está ideado para permitir establecer, modificar y finalizar sesiones con más de un participante. Recientemente se han detectado CVE (*Common vulnerabilities and exposures*) vulnerabilidades y exposiciones a riesgos que precisamente, este preprocesador detecta con facilidad.

EL procesador ISP requiere que Stream5 guarde las sesiones tanto para UDP como para TCP. Además Stream API es recomendable para trabajar con canales de audio y video. Su configuración establece en máximos de sesiones seguidas (`max_sessions`), los puertos (`ports`), la lista de métodos a analizar (`methods\{\}),`el ignorar canales de audio/video con Stream API (`ignore_call_channel`).

- Ejemplo de configuración:

```
preprocessor sip: ports { 5060 49848 36780 10270 },
\ max_call_id_len 200, \
max_from_len 100, max_to_len 200, max_via_len 1000, \
max_requestName_len 50, max_uri_len 100, ignore_call_channel,\
max_content_len 1000
```

A.18. Reputation

Este preprocesador permite establecer listas que enmarquen la reputación de determinadas direcciones IP. Permitirá a Snort trabajar con listas blancas y negras de direcciones y establecerá si el tráfico involucrado con éstas pasará, se bloqueará o será descartado.

- En el ejemplo vemos como referencia archivos de listas blancas y negras:

```
preprocessor reputation: \
memcap 4095, scan_local, nested_ip both, \
priority whitelist, \
blacklist /etc/snort/default.blacklist, \
whitelist /etc/snort/default.whitelist
```

El parámetro `nested_ip`indica la dirección a usar cuando la IP sea encapsulada, podemos establecer la prioridad (`priority`), el tamaño máximo permitido de memoria (`memcap),`si inspeccionar direcciones locales (`nested_ip`).

A.19. Descodificador y Preprocesador GTP

El protocolo de túnel de GPRS (GTP) es un grupo de protocolos de comunicaciones basados en IP que se usan para portar el servicio GPRS dentro de las redes GSM y UMTS. El protocolo GTP se puede descomponer en varios protocolos independientes, GTP-C, GTP-U y GTP:

1. El protocolo GTP-C se usa en la red GPRS para señalización entre el Nodo de Soporte del Servicio GPRS (SGSN) y el Nodo de Soporte de la Compuerta GPRS (GGSN). Este le permite al SGSN activar una sesión de usuario (activación del contexto PDP), para desactivar la misma sesión, ajustar los parámetros de calidad de servicio, o actualizar una sesión para un abonado que acabe de llegar de otro SGSN.

2. El protocolo GTP-U se usa para portar datos de usuario dentro de la red GPRS y la Red de Acceso de Radio (RAN) y la red GSM. Los datos de usuario transportados pueden estar los formatos de paquetes IPv4, IPv6 y PPP.

3. El protocolo GTP' (GTP prima) usa la misma estructura de mensaje del GTP-C y GTP-U, pero tiene una función independiente. Este puede usarse para portar datos de tasación desde la función de tasación (CDF) de la red GSM o red UMTS hasta la función de compuerta de tasación (CGF). Esto generalmente quiere decir, desde varios elementos individuales de la red tales como el GGSN hasta el computador centralizado que proporciona los datos de tasación al centro de facturación del operado

Este preprocesador está diseñado para evitar técnicas de ataque basadas en estos protocolos. Requiere del uso de Stream5 para guardar las sesiones tanto UDP como IP. Además Frag3 debe estar activado en modo IP.

A.20. Modbus

Modbus es un protocolo de comunicaciones situado en el nivel 7 del Modelo OSI, basado en la arquitectura maestro/esclavo o cliente/servidor, diseñado en 1979 por Modicon para su gama de controladores lógicos programables (PLCs). Convertido en un protocolo de comunicaciones estándar de facto en la industria es el que goza de mayor disponibilidad para la conexión de dispositivos electrónicos industriales.

Modbus permite el control de una red de dispositivos, por ejemplo un sistema de medida de temperatura y humedad, y comunicar los resultados a un ordenador. Modbus también se usa para la conexión de un ordenador de supervisión con una unidad remota (RTU) en sistemas de supervisión adquisición de datos (SCADA). Existen versiones del protocolo Modbus para puerto serie y Ethernet (Modbus/TCP).

La función de este preprocesador es facilitar la descodificación de dicho protocolo mediante el paso de reglas específicas. De hecho, si la red no dispone de dispositivos Modbus es recomendable deshabilitarlo para ganar eficiencia. El preprocesador Modbus requiere de Stream5 guardando sesiones TCP, el protocolo PAF (*Protocol Aware Flushing*) y Frag3 con desfragmentación IP.

A.21. DNP3

DNP3 (acrónimo del inglés *Distributed Network Protocol*, en su versión 3) es un protocolo industrial para comunicaciones entre equipos inteligentes (IED) y estaciones controladores, componentes de sistemas SCADA. Es un protocolo ampliamente utilizado en el sector eléctrico, de gran difusión en Estados Unidos y Canadá, y menor presencia en Europa donde el uso de alternativas como IEC-60870 101 e IEC-60870 104 gozan de mayor popularidad. También se puede encontrar en otros campos (agua, gas, entre otros tipos de empresas de servicio).

El objetivo de este preprocesador es decodificar este tipo de tráfico para facilitar la tarea del detector. Requiere de Stream5 guardando sesiones sobre TCP o UDP, PAF(*Protocol Aware Flushing*) y Frag3 en modo IP.

Apéndices B

Alertas de Snort

Snort dispone de 11 modos distintos para generar las alertas. La elección de un formato u otro depende de los detalles que necesitemos o de cómo deseemos visualizar esas alertas. La configuración de los módulos de salida se hace con el archivo de configuración snort.conf. Al abrir este fichero tenemos que buscar la sección output plug-ins. Por defecto esta sección está etiquetada con el comentario Step #3: Configure output plug-ins. Los distintos modos que podemos usar son:

B.1. Alert_syslog:

Envía las alarmas al syslog. De esta manera si una computadora es atacada los registros no se ven comprometidos, puesto que fueron mandados a otro sistema. Para que Snort envíe las alertas a un registro de sistema tendremos que añadir la opción -s a nuestra lista de comandos Snort y modificar el fichero de configuración *snort.conf*.

```
###################################################################
# Step #4: Configure output plugins
#
# Uncomment and configure the output plugins you decide to use.  General
# configuration for output plugins is of the form:
#
# output <name_of_plugin>: <configuration_options>
#
# alert_syslog: log alerts to syslog
# ================================
# Use one or more syslog facilities as arguments.  Win32 can also optionally
# specify a particular hostname/port.  Under Win32, the default hostname is
# '127.0.0.1', and the default port is 514.
#
# [Unix flavours should use this format...]
# output alert_syslog: LOG_AUTH LOG_ALERT
#
# [Win32 can use any of these formats...]
# output alert_syslog: LOG_AUTH LOG_ALERT
# output alert_syslog: host=hostname, LOG_AUTH LOG_ALERT

# output alert_syslog: host=hostname:port, LOG_AUTH LOG_ALERT

output alert_syslog: host=192.168.1.30:514, LOG_LOCAL7 LOG_ALERT LOG_CONS LOG_NDELAY LOG_PERROR LOG_PID
```

Figura B.1: Cambios en Snort.conf

Para terminar, en la línea de comandos para ejecutar Snort:

```
snort -i3 -A console -dev -l C:\Snort\log -h 192.168.101.240/24 -c C:\Snort\etc\snort.con
```

Donde:

-s activamos el envio a syslog

192.168.1.30 máquina donde corre el syslog

514 puerto UDP donde escucha el servicio syslog

A continuación vemos una captura de alertas de snort en una máquina remota:

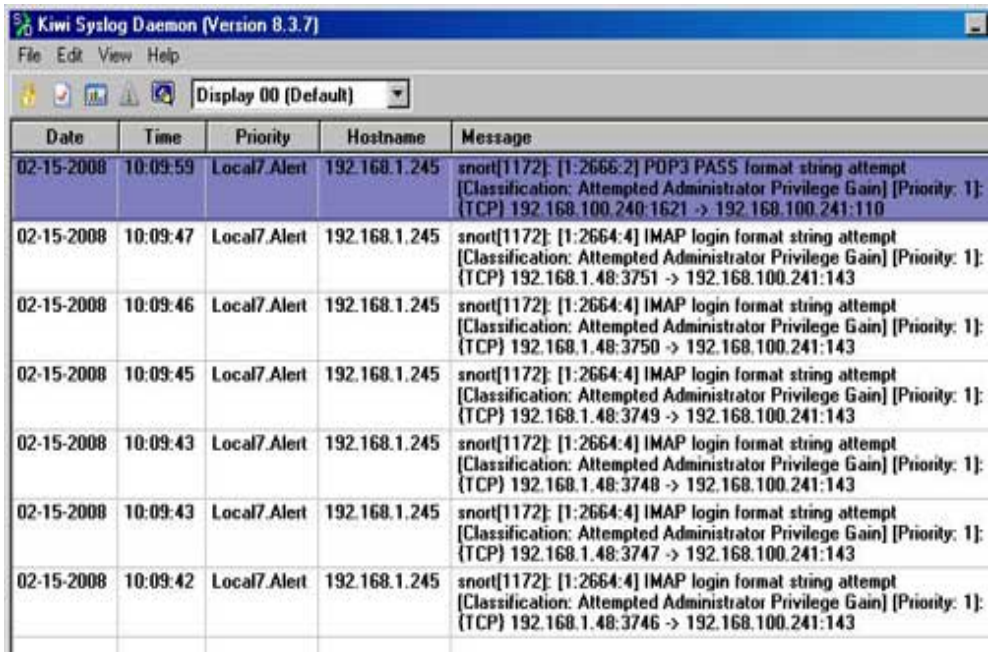

Figura B.2: Captura de alertas Snort en máquina remota

B.2. Alert_fast:

El modo alerta rápida devuelve información sobre tiempo, el propio mensaje de alerta , prioridad, IP , y puertos en formato ASCII. Se escriben las alertas en formato de una línea en el fichero que especifique el usuario. Es un modo muy rápido puesto que Snort no desperdicia capacidad de procesamiento haciendo conversiones de formato.

- Para utilizar este modo deberemos escribir en el fichero snort.config la siguiente línea:

  ```
  output alert_fast: snort.log
  ```

- Para utilizar este modo al ejecutar Snort por consola debemos añadir la opción **−A fast**. Las alertas generadas en este formato contienen la siguiente información: fecha y hora, mensaje de la alerta (núcleo), clasificación, prioridad de la alerta, protocolo, IP y puertos de origen y destino.

```
snort -A fast -dev -l ./log -h 192.168.4.0/24 -c ../etc/snort.conf
09/19-19:06:37.421286 [**] [1:620:2] SCAN Proxy (8080) attempt [**]
[Classification: Attempted Information Leak] [Priority: 2] ...
... {TCP} 192.168.4.3:1382 -> 192.168.4.15:8080
```

- Ejemplo de alerta:

  03/02-11:04:35.256648 marca de tiempo

  1:382:7 numeración asociada a la descripción de la alerta

  ICMP PING Windows nombre de la alerta

  Classifications: Misc activity clasificación de la alerta contenida en el archivo classification.config.

  Priority: 3 prioridad de la alerta

  ICMP protocolo asociado a la generación de la alerta

  192.168.1.5 origen que genera la alerta

  192.168.1.239 destino de quien genera la alerta

B.3. Alert_full:

El modo de Alerta Completa nos devolverá información sobre: tiempo, mensaje de la alerta, clasificación, prioridad de la alerta, IP y puerto de origen / destino e información completa de las cabeceras de los paquetes registrados en Formato ASCII.

Si seleccionamos el modo Full obtendremos, además de la información del formato Fast, las cabeceras de los paquetes que 102 generan alertas. Este modo puede hacer que Snort no analice todos los paquetes en redes con mucho tráfico al no poder procesarlos. Debido a esto es un modo poco recomendable incluso en redes con poco tráfico.

- La sintaxis a utilizar en el fichero snort.config para utilizar el modo 1 alert_full es la siguiente:

  ```
  output alert_full: alert.full
  ```

- Para utilizar este modo al ejecutar Snort por consola debemos añadir la opción -A full. En el siguiente ejemplo podemos ver cómo se mostrarán los ataques en este formato:

  ```
  snort -A full -dev -l ./log -h 192.168.4.0/24 -c ../etc/snort.conf
  [**] [1:620:2] SCAN Proxy (8080) attempt [**]
  [Classification: Attempted Information Leak] [Priority: 2]
  09/19-14:53:38.481065 192.168.4.3:3159 -> 192.168.4.15:8080
  TCP TTL:128 TOS: 0x0 ID:39918 IpLen:20 DgmLen:48 DF
  ******S* Seq: 0xE87CBBAD Ack: 0x0 Win: 0x4000 TcpLen: 28
  TCP Options (4) => MSS: 1456 NOP NOP SackOK
  ```

- Ejemplos:

1. Información de la cabecera del paquete:

```
TCP TTL:128 TOS:0x0 ID:39918 IpLen:20 DgmLen:48 DF
******S* Seq: 0E87CBBAD Ack: 0x0 Win: 0x4000 TcpLen: 28
TCP Options (4) => MSS: 1456 NOP NOP SackOK
```

2. Añadimos en esta ocasión el formato tcpdump.

```
1:382:7
ICMP PING Windows
Classifications: Misc Activity
Priority: 3 prioridad de la alerta
```

3. Esta parte que viene a continuación correspondería con Frame Wireshark de que nos muestra información completa de la trama capturada. Tamaño total, etc. y Ethernet:

```
03/02-11:06:05.314380
0:4:75:ED:89:DF -> 0:15:C5:89:85:5B
type:0x800
len:0x4A
```

4. Esta parte corresponde a Internet Protocol (IP) e Internet Control Message Protocol (ICMP) con información de los campos de las cabeceras:

```
192.168.1.5 -> 192.168.1.239
ICMP
TTL:128
TOS:0x0
ID:56307
IpLen:20
DgmLen:60
Type:8   Code:0   ID:1280     Seq:12288   ECHO
```

B.4. Alert_unixsock:

Manda las alertas a través de un socket para que las escuche otra aplicación. Está opción solo puede usarse en GNU Linux y OSX. Experimental.

B.5. Log_tcpdump:

Se registran los datos en formato binario (tcpdump-formatted file). Esto permite que Snort se ejecute más rápido (no se pierde tiempo de conversión de datos). Este modo es especialmente útil cuando deseamos hacer un análisis posterior del tráfico de una red. Con los datos en formato tcpdump podemos utilizar cualquier utilidad de procesamiento. Para usar este modo de alertas escribiremos en el archivo de configuración la siguiente línea:

`outputlog_tcpdump:snort.dump`.

Si se desea registrar todos los paquetes debe usarse la opción –b. Con la opción-L especificamos el destino:

`/usr/local/bin/snort{b {L/var/log/snort/snort.dump`.

B.6. Database:

Este modo guarda el log de alertas en una base de datos con las ventajas que ello conlleva: mayor velocidad y facilidad para buscar los archivos. La base de datos se puede configurar de manera que solamente almacene alertas, logs o las dos cosas. Snort solo soporta bases de datos que usen MySQL. Para usar este modo incluiremos en el archivo de configuración la siguiente línea:

`outputdatabase:<log|alert>,<databasetype>,<parameter`list¿.

Los parámetros que deberemos incluir en el fichero de configuración snort.conf se especifican con el par `name=value`y son los siguientes:

- **Host:** Nombre o dirección IP del host el que se está ejecutando el servidor de la base de datos. Si la base de datos está en el mismo sistema donde está Snort el nombre del host será localhost.
- **Port:** Puerto de escucha del servidor de la base de datos.
- **Password:** Contraseña con la que se accede a la base de datos (opcional).
- **Dbname** Nombre de la base de datos.
- **Encoding:** Método de codificación de los paquetes (Hex, Base64 o Ascii).
- **Sensor_name:** Nombre del sensor de Snort (generalmente no es necesario especificarlo.
- **Detail:** Nivel de detalle con el que se almacenan los paquetes (full o fast)
- Una posible configuración podría ser:

```
output database: log, mysql, user=snortuser \
password=h4wg dbname=snortdb host=localhost
```

- Ejemplo:

```
# database: log to a variety of databases
# |||||||||||||
# See the README.database file for more information about configuring
# and using this plugin.
#
# output database: log, mysql, user=root password=test
# dbname=db host=localhost
# output database: alert, postgresql, user=snort dbname=snort
# output database: log, odbc, user=snort dbname=snort
# output database: log, mssql, dbname=snort user=snort password=test
# output database: log, oracle, dbname=snort user=snort password=test
```

La base de datos se suele crear en la instalación de Snort. En caso contrario se puede crear manualmente mediante los siguientes pasos:

1. Creación de la base de datos de Snort

```
mysql -u root -p
> create database snort;
> exit
```

2. Ahora vamos a ejecutar el script create_mysql para crear las tablas que serán usadas por snort:

```
mysql -u root -p -D snort < c:\Snort\schemas\create_mysql
Enter password: **************
mysql -u root -p
Enter password: **************
```

3. Y por último le otorgamos permisos:

```
> grant insert,select,update,create,delete on snort.* \
to snort@localhost identified by 'snortpass';
>exit
```

B.7. CSV:

CSV es un formato intermedio universal para exportar datos en una base de datos. Se utiliza cuando se está utilizando una base de datos que no es soportada por Snort.

- Ejemplo:

```
output alert_csv: /var/log/alert.csv default
output alert_csv: /var/log/alert.csv timestamp, msg
```

En el primer caso mostrará los campos por defecto, y el el segundo el timestamp y msg. La lista de campos a mostrar: timestamp, sig generator, sig id, sig rev, msg, proto, src, etc. También podemos fijar el límite de almacenamiento.

B.8. Unified:

Este modo está diseñado para ser uno de los más rápidos, de manera que los registros se almacenan en formato binario. Unified genera dos archivos: uno de alertas con información de las condiciones en que se añadió una nueva entrada y otro log con la información de los paquetes. El archivo `spo_unifieed.h`describe como se compone dicho archivo binario.

- Ejemplo:

```
output alert_unified: snort.alert, limit 128
output log_unified: snort.log, limit 128
```

B.9. Unified2:

Unified2 es una versión ampliada del modo Unified. Pudiendo ejecutarse en 3 modos distintos: packet logging , alert logging y true unified logging.

- Ejemplo:

```
output alert_unified2: filename snort.alert, limit 128, nostamp
output log_unified2: filename snort.log, limit 128, nostamp
output unified2: filename merged.log, limit 128, nostamp
output unified2: filename merged.log, limit 128, nostamp, mpls_event_types
```

B.10. Alert_prelude:

Este modo de ejecución se utiliza para registrar los log en una base de datos de Prelude (para obtener información sobre Prelude visitar la página web http://www.prelude-ids.org/). A partir de la versión 2.9.3 de Snort ha sido eliminado.

- Ejemplo:

```
output alert_prelude: profile=snort info=4 low=3 medium=2
```

B.11. Log_null:

Usando este modo de ejecución se desactivara el registro de paquetes. Para lanzarlo usamos el siguiente comando:

- Ejemplo:

```
snort -A none -c snort.conf
```

B.12. Alertas en función del formato

Snort dispone de otros 3 tipos de alertas clasificadas en función del formato: pcap, ascii, none. Tanto el formato ascii como pcap son enviados y guardados por Snort en el logdir o carpeta de registros ubicada normalmte en snort/log que establecemos con el comando -l.

El formato pcap puede ser interpretado por analizadores tales como Windump/Tcpdum o Wireshark para su mejor comprensión y análisis. Desde Wireshark tan solo tendremos que ir a File ¿Open y abrir el archivo pcap del volcado de la alerta. Desde Windump/Tcpdump y con la opción -r leemos el archivo de formato pcap.

El formato ascii es el formato antiguo de logging y bastante interesante ya que diferencia el volcado de las alertas atendiendo a las direcciones IP causantes de dichas alertas. Se establece entonces una estructura de directorios o carpetas en /log con las direcciones IP

.Dentro de cada carpeta tenemos los volcados de las alertas correspondientes en formato ascii con todo el contenido de los paquetes. Cualquiera de estos archivos se abre con el Bloq de notas, Notepad o similar.

Los tres modos de alertas los podemos establecer con -K

- **-K** pcap (por defecto)
- **-K** ascii
- **-K** none

Apéndices C

Librería OpenMP

C.1. Hilos

Un *thread* es un hilo de ejecución. Un proceso puede consistir de múltiples threads, cada uno con su propio flujo de control pero compartiendo el mismo espacio de direcciones. Todo programa iniciará con un hilo principal del que pueden iniciarse nuevos hilos que se ejecutarán en paralelo y que pueden ser recogidos al final. La siguiente ilustración muestra el esquema típico de una aplicación *multithread*. Cada hilo realizará una labor cuyo producto será recogido al final. Esto se puede apreciar en la Figura C.1

Al ser OpenMP es un modelo de memoria compartida, los threads se comunican utilizando variables compartidas. El uso inadecuado de variables compartidas origina carreras críticas, es por esto que para controlar las carreras críticas, se requiere el uso de sincronización para protegerse de los conflictos de datos. La sincronización es costosa, entonces es útil modificar cómo se almacenan los datos para minimizar la necesidad de sincronización.

C.1.1. Cómo aprovechar el TLP (*Thread Level Paralelism*)

OpenMP principalmente explota el paralelismo a nivel de bucles (grano fino) y regiones paralelas(grano grueso). En el paralelismo a nivel de bucle se le asignará un único indice a cada *thread*.

En el caso de paralelizar regiones paralelas cualquier sección del código puede ser paralelizado. Las caracteristicas de los algoritmos implementados en nuestro NIDS permiten explotar con relativa facilidad el paralelismo a nivel de bucle. Precisamente son dichas iteraciones las que penalizan el tiempo de ejecución del programa en las zonas críticas, es decir, de mayor coste de ejecución, y nos centraremos en ellas para tratar de optimizar el rendimiento en tiempo real del sistema.

C.1.2. Directivas OpenMP

Se trata de directivas de compilación que adecuadamente utilizadas permiten explotar el paralelismo implícito de los algoritmos. Cada directiva inicia con `#pragma omp` .

```
#pragma omp directive-name[clause[,] clause]...
```

Se trata de directivas de compilación que adecuadamente utilizadas permiten explotar el paralelismo implícito de los algoritmos. Cada directiva inicia con `#pragma omp`.

- **Constructor Paralelo:** Las siguientes directivas definen una región paralela, la cual es una parte del programa que puede ser ejecutada por múltiples hilos en paralelo. Este es el constructor fundamental que inicia la ejecución paralela.

  - **Ejemplo**

    ```
    #pragma omp parallel [clause[,] clause]...
    {
    bloque de código
    }
    ```

  - **Clausulas:**

    Private(lista de variables La lista de variables puede ser separada por .,. y para cada una de ellas se genera una copia por hilo. La copia no tiene relación con la original y no es inicializada a menos que se utilice la opción `firstprivate`.

    Shared(lista de variables) Las variables de la lista son comunes a todos los hilos y cada uno de ellos puede modificarlas con efecto global.

    Threadprivate(lista de variables) Hace que la lista sea privada a cada hilo, pero será global dentro dicho hilo.

    Copyin(lista de variables) Hace que al comienzo de una región paralela la lista de parámetros del hilo maestro se copie en las variables privadas del resto de hilos.

    Reduction(operador:lista de variables) Realiza una operación de reducción sobre las variables que aparecen en la lista utilizando el operador/intrínseco especificado.

    - El operador puede ser: `+, *, ., &(and), |(or), ^(eqv), &&(neqv)`.
    - El intrínseco puede ser: `||(max), (min), (iand), (ior)`.

- **Constructores de trabajo Compartido**

  - **Constructor for :** La directiva for identifica un constructor de trabajo compartido el cual especifica que las iteraciones del loop asociado deben ser ejecutadas en paralelo.

    ```
    #pragma omp for[clause[,] clause]...
    {
    {bloque de código }
    }
    ```

  - **Constructor Sections:** En la directiva sections cada sección es ejecutada una vez por un hilo en el grupo. La sintaxis de la directiva sections es la siguiente:

    ```
    #pragma omp sections[clause[,] clause]...
    {
    #pragma omp section
    {bloque de código }
    #pragma omp section
    {bloque de código }
    }
    ```

- **Constructor Single:** Esta directiva se utiliza para referirse a que en cada tramo del código va a ejecutarse por un solo hilo. La sintaxis de la directiva single es la siguiente:

```
#pragma omp single[clause[,] clause]...
{
{bloque de código}
}
```

- **Constructores de Sincronización:**

  - **Constructor "master"** La directiva `for` identifica un constructor de trabajo compartido el cual especifica que las iteraciones del bucle asociado deben ser ejecutadas en paralelo.

```
#pragma omp master
{
{bloque de código}
}
```

  - **Constructor "critical":** En la directiva sections cada sección es ejecutada una vez por un hilo en el grupo. La sintaxis de la directiva sections es la siguiente:

```
#pragma omp critical [(nombre)]
{
{bloque de código}
}
```

  - **Constructor "barrier":** Esta directiva se utiliza para referirse a que en cada tramo del código va a ejecutarse por un solo hilo. La sintaxis de la directiva single es la siguiente:

```
...
#pragma omp barrier
```

  - `Constructor \atomic":` Este constructor especifica que el bloque de código es ejecutado únicamente por el hilo maestro.

```
#pragma omp atomic
```

  - **Constructor "flush":** Este constructor indica que el bloque de código es accedido por un solo hilo a la vez. Cada hilo espera el comienzo de bloque hasta se libere.

```
...
#pragma omp flush(lista de variables)
```

C.2. Funciones OpenMP:

En barrier cada hilo espera hasta que cada uno de los demás hilos termine.

- **omp_set_num_threads:** Establece el número de hilos que se emplearán hasta nueva indicación.
- **omp_get_num_threads:** Devuelve el número de hilos actuales.
- **omp_get_max_threads:** evuelve el Máximo numero de hilos.
- **omp_get_threads_num:** Número de hilos en que se ejecuta el presente thread. Master se considera hilo 0.
- **omp_get_num_procs:** evuelve el número de procesadores a usar.
- **omp_in_parallel:** devuelve un valor diferente de cero si es llamada desde una extensión dinámica de una región paralela ejecutada en paralelo. En otro caso devuelve un valor de cero.
- **omp_get_dynamic:** sta función devuelve un valor diferente de cero si el ajuste de hilos dinámico es activo. En caso contrario devuelve cero.
- **omp_set_dynamic:** ctiva o desactiva el ajuste dinámico del número de hilos disponible para la ejecución de una región en paralelo.
- **omp_set_nested:** activa o desactiva paralelismo anidado.
- **omp_get_nested:** devuelve un valor diferente de cero si el paralelismo anidado esta activo y cero en caso contrario.

C.3. Variables de entorno OpenMP

Algunas de las variables de entorno que ofrece OpenMp:

- **OMP_SCHEDULE**: Especifica como se va a distribuir el trabajo en cada uno de los hilos. Ésta función solo se utiliza para los constructores "for" o "parallel for" de manera dinámica o estática.
- **OMP_NUM_THREADS**: Asigna la cantidad de hilos que se van a utilizar durante la ejecución. Si se hace un llamado a función: `omp_set_num_thread` el número de hilos se cambia.
- **OMP_DYNAMIC**: Activa o desactiva un ajuste dinámico del número de hilos disponibles en una ejecución loop o de una región paralela.
- **OMP_NESTED**: Activa o desactiva el paralelismo anidado excepto a excepción de que sea redefinido por `omp_set_nested`.

C.4. Ejemplos

1. Paralelización de loop.

   - Programa secuencial

     ```
     void main() {
     double a[1000],b[1000],c[1000];
     for (int i = 0; i< 1000; i++){
     a[i] = b[iI] + c[i];
     }
     }
     ```

   - Programa Paralelo

     ```
     void main() {
     double a[1000],b[1000],c[1000];
     #pragma omp parallel for
     for (int i = 0; i< 1000; i++){
     a[i] = b[iI] + c[i];
     a[i] = b[iI] + c[i];
     }
     }
     ```

2. Regiones paralelas: el funcionamiento de este código se muestra en la Figura C.2.

   ```
   i = 0;
   omp_set_num_threads(4);
   #pragma omp parallel
   {
   i = omp_thread_num();
   foo(i,a,b,c);
   }
   #pragma omp end parallel
   printf(\%d\n", i);
   ```

3. Ejemplo de cómo compartir trabajo:

   - Secuencial

     ```
     for(i=0;i<n;i++) {
     a[i] = a[i] + b[i];
     }
     ```

   - Region Paralela OpenMP

     ```
     #pragma omp parallel
     {
     Int id, i, Nthreads, istart, iend;
     Id = omp_get_thread_num(i);
     Nthreads = omp_get_num_threads();
     Istart = id * N / Nthreads;
     Iend = (id + 1) * N / Nthreads;
     ```

```
For(i=istart;i<iend;i++) {a[i]=a[i]+b[i];}
}
```

- Region Paralela y Constructor para repartir trabajo OpenMp

```
#pragma omp parallel
#pragma omp for schedule(static)
for(i=0;i<n;i++) {a[i] = a[i] + b[i];}
```

4. Ejemplo de exclusión mutua

```
#pragma omp parallel shared(x,y)
...
#pragma omp critical (section1)
actualiza(x);
#pragma omp end critical(section1)
...
#pragma omp critical(section2)
actualiza(y);
#pragma omp end critical(section2)
#pragma omp end parallel
```

5. Ejemplo de calculo de Pi secuencial

```
#include <omp.h>
static long num_steps = 100000; double step;
#define NUM_THREADS 2
void main ()
{ int i; double x, pi, sum[NUM_THREADS];
step = 1.0/(double) num_steps;
omp_set_num_threads(NUM_THREADS);
#pragma omp parallel
{ double x; int id;
id = omp_get_thread_num();
for (i=id, sum[id]=0.0;i< num_steps; i=i+NUM_THREADS){
x = (i+0.5)*step;
sum[id] += 4.0/(1.0+x*x);
}
}
for(i=0, pi=0.0;i<NUM_THREADS;i++) pi += sum[i] * step;
}
```

6. Calculo de Pi compartiendo trabajo

```
#include <omp.h>
static long num_steps = 100000; double step;
#define NUM_THREADS 2
void main ()
{ int i; double x, pi, sum[NUM_THREADS];
step = 1.0/(double) num_steps;
omp_set_num_threads(NUM_THREADS)
```

```
#pragma omp parallel
{ double x; int id;
id = omp_get_thread_num(); sum[id] = 0;
#pragma omp for
for (i=id;i< num_steps; i++){
x = (i+0.5)*step;
sum[id] += 4.0/(1.0+x*x);
}
}
for(i=0, pi=0.0;i<NUM_THREADS;i++)pi += sum[i] * step;
}
```

7. Calculo de Pi utilizando Reduction

```
#include <omp.h>
static long num_steps = 100000; double step;
#define NUM_THREADS 2
void main ()
{ int i; double x, pi, sum = 0.0;
step = 1.0/(double) num_steps;
omp_set_num_threads(NUM_THREADS);
#pragma omp parallel for reduction(+:sum) private(x)
for (i=1;i<= num_steps; i++){
x = (i-0.5)*step;
sum = sum + 4.0/(1.0+x*x);
}
pi = step * sum;
}
```

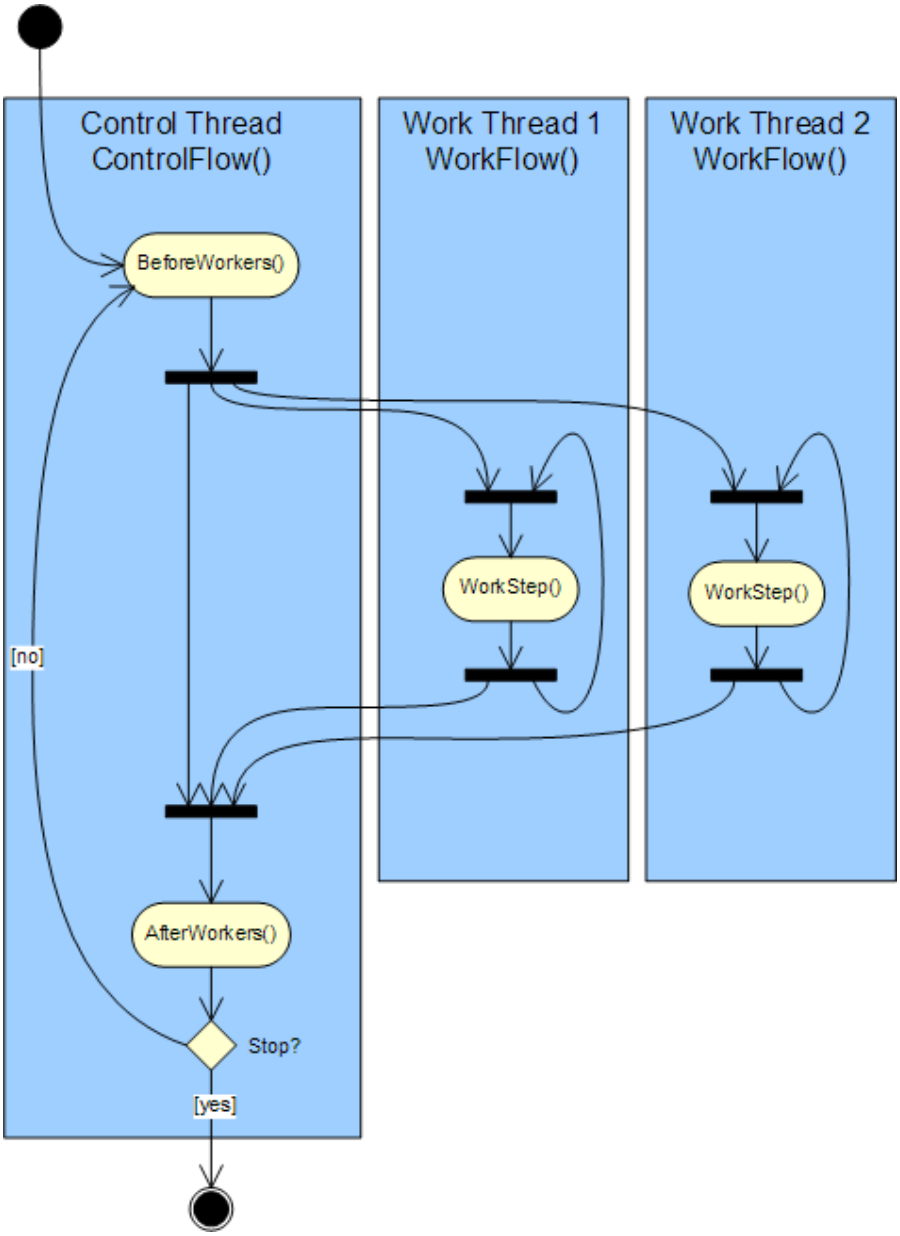

Figura C.1: Esquema funcionamiento paralelización OpenMP

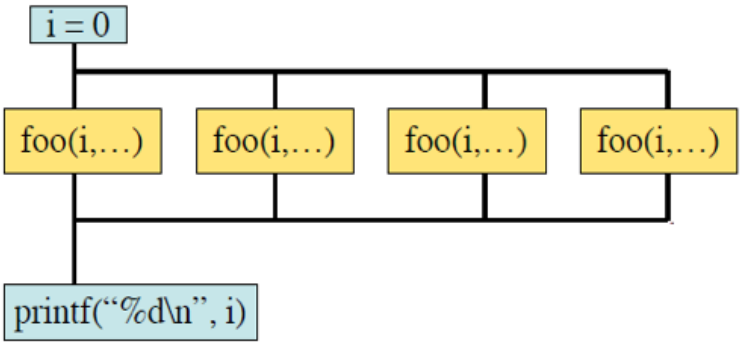

Figura C.2: Esquema regiones paralelas OpenMP